

版权信息

书名：物联网设计：从原型到产品

作者：[英] Adrian McEwen Hakim Cassimally

译者：张崇明

ISBN：978-7-115-37641-1

本书由北京图灵文化发展有限公司发行数字版。版权所有，侵权必究。

您购买的图灵电子书仅供您个人使用，未经授权，不得以任何方式复制和传播本书内容。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。

图灵社区会员 人民邮电出版社（zhanghaichuan@ptpress.com.cn） 专享
尊重版权

版权声明

译者序

致谢

引言

 本书的局限

 本书的目标读者

 怎样使用本书

 写作手记

第一部分 开发阶段

第 1 章 物联网概览

 1.1 物联网的应用范例

 1.2 物联网的概念

 1.3 物联网出现的技术背景

 1.4 有魔力的物品

 1.5 物联网的创造者

 1.6 小结

第 2 章 联网装置的设计原则

 2.1 环境计算和宁静技术

 2.2 用魔法作隐喻

 2.3 隐私

 2.3.1 保守秘密

 2.3.2 谁的数据

 2.4 联网装置的Web思维

 2.4.1 小块松散组合

 2.4.2 因特网上的一等公民

 2.4.3 优雅降级

 2.5 功能可供性

 2.6 小结

第 3 章 因特网原理

3.1 因特网通信概览

3.1.1 IP

3.1.2 TCP

3.1.3 IP协议栈

3.1.4 UDP

3.2 IP地址

3.2.1 DNS

3.2.2 静态IP地址分配

3.2.3 动态IP地址分配

3.2.4 IPv6

3.3 MAC地址

3.4 TCP和UDP端口

3.4.1 示例：HTTP端口

3.4.2 其他常用端口

3.5 应用层协议

3.5.1 HTTP

3.5.2 HTTPS：加密的HTTP

3.5.3 其他应用层协议

3.6 小结

第 4 章 原型设计与制作概述

4.1 快速搭建原型

4.2 熟悉程度

4.3 成本与开发难度

4.4 原型和产品

4.4.1 修改嵌入式平台

4.4.2 原型结构和批量个性化定制

4.4.3 迁移到云端

4.5 开源与闭源

4.5.1 为何选择闭源

4.5.2 为何选择开源

4.5.3 混合使用开源和闭源

4.5.4 在大众市场项目中选择闭源

4.6 利用社区资源

4.7 小结

第 5 章 嵌入式装置的原型开发

5.1 电子电路基础

5.1.1 传感器

5.1.2 执行器

5.1.3 原型电路的演进路线

5.2 嵌入式计算基础

5.2.1 微控制器

5.2.2 片上系统

5.2.3 选择平台

5.3 Arduino

5.3.1 在Arduino上做开发

5.3.2 硬件相关的一些介绍

5.3.3 开放性

5.4 树莓派

5.4.1 外壳和扩展板

5.4.2 用树莓派做开发

5.4.3 硬件相关的一些说明

5.4.4 开放性

5.5 BeagleBone Black

5.5.1 外壳和扩展板

5.5.2 在BeagleBone上做开发

- 5.5.3 硬件相关的一些说明
 - 5.5.4 开放性
- 5.6 Electric Imp
- 5.7 其他值得关注的平台
 - 5.7.1 手机和平板电脑
 - 5.7.2 插头计算：始终在线的物联网
- 5.8 小结
- 第 6 章 原型系统的结构与制作
 - 6.1 准备工作
 - 6.2 画草图，迭代和探索
 - 6.3 非数字化的方法
 - 6.4 激光切割
 - 6.4.1 激光切割机的选择
 - 6.4.2 软件
 - 6.4.3 铰链和接头
 - 6.5 3D打印
 - 6.5.1 3D打印技术的类型
 - 6.5.2 软件
 - 6.6 数控铣削
 - 6.7 现有物品的循环再利用
 - 6.8 小结
- 第 7 章 原型系统在线组件的设计
 - 7.1 开始使用API
 - 7.1.1 API的混聚
 - 7.1.2 Web数据抓取
 - 7.1.3 合法性
 - 7.2 编写新的API
 - 7.2.1 Clockodillo

- 7.2.2 安全
 - 7.2.3 API的实现
 - 7.2.4 使用CURL进行测试
 - 7.2.5 进一步的工作
 - 7.3 实时响应
 - 7.3.1 轮询
 - 7.3.2 COMET
 - 7.4 其他协议
 - 7.4.1 消息队列遥测传输
 - 7.4.2 可扩展通信和表示协议
 - 7.4.3 受限应用协议
 - 7.5 小结
- 第 8 章 嵌入式编程技术
- 8.1 内存管理
 - 8.1.1 内存类型
 - 8.1.2 最大程度地利用RAM
 - 8.2 性能和电池寿命
 - 8.3 库
 - 8.4 调试
 - 8.5 小结
- 第二部分 产品阶段
- 第 9 章 商业模式
- 9.1 商业模式简史
 - 9.1.1 空间和时间
 - 9.1.2 从手工制作到批量生产
 - 9.1.3 因特网时代的长尾效应
 - 9.1.4 以史为鉴
 - 9.2 商业模式画布

9.3 商业模式的用途

9.4 常见模式

9.4.1 制造销售

9.4.2 订阅

9.4.3 定制化

9.4.4 成为一种关键资源

9.4.5 提供基础设施：传感器网络

9.4.6 获取提成

9.5 为物联网初创企业筹资

9.5.1 业余爱好项目和开源

9.5.2 风险投资

9.5.3 政府投资

9.5.4 众筹

9.6 精益创业

9.7 小结

第 10 章 生产制造阶段

10.1 你要生产什么

10.2 设计套件

10.3 设计印制电路板

10.3.1 软件选择

10.3.2 设计过程

10.4 制作印制电路板

10.4.1 蚀刻电路板

10.4.2 电路板的铣加工

10.4.3 第三方制作

10.4.4 装配

10.4.5 测试

10.5 批量生产壳体和其他固定物

10.6 认证

10.7 成本

10.8 扩展软件

10.8.1 部署

10.8.2 正确性和可维护性

10.8.3 安全

10.8.4 性能

10.8.5 用户社区

10.9 小结

第 11 章 道德伦理

11.1 描述物联网的特征

11.2 隐私

11.3 控制

11.3.1 混乱的控制

11.3.2 众包

11.4 环保

11.4.1 实体装置

11.4.2 电子电路

11.4.3 因特网服务

11.5 解决之道

11.5.1 把物联网作为解决方案的一部分

11.5.2 谨慎乐观

11.5.3 开放物联网的定义

11.6 小结

版权声明

All Rights Reserved. This translation published under license. Authorized translation from the English language edition, entitled *Designing the Internet of Things*, ISBN 9781118430620, by Adrian McEwen and Hakim Cassimally, Published by John Wiley & Sons. No part of this book may be reproduced in any form without the written permission of the original copyrights holder.

Simplified Chinese translation edition published by POSTS & TELECOM PRESS Copyright © 2015.

本书简体中文版由John Wiley & Sons, Inc.授权人民邮电出版社独家出版。

本书封底贴有John Wiley & Sons, Inc.激光防伪标签，无标签者不得销售。

版权所有，侵权必究。

译者序

随着纪录片《互联网时代》的热播，很多人都在微博上热议这个话题，开始思考自己应该怎样适应这个新的时代，怎样把握其中的机会。而随着越来越多的物品开始具备联网能力并被接入互联网，物联网时代也正在到来。为了应对物联网时代的新挑战，把握新的机遇，你可能觉得有必要先人一步去了解一下物联网。

虽然市面上已经可以看到不少物联网相关的书籍，但与之相比，本书的特色十分鲜明。主要体现在以下两个方面。

首先，本书的内容十分全面。本书包括“原型阶段”和“产品阶段”两大部分内容。在“原型阶段”对联网装置的设计原则的讨论，对装置原型的结构设计（包括对激光切割、3D打印和数控铣加工）的介绍，在“产品阶段”对商业模式、生产制造过程和道德伦理的介绍和讨论，都是大多数物联网相关的技术书籍所不具备的。

其次，本书的内容非常实用，且写作思路很清晰。作者从帮助你产生好的想法开始，到制作产品原型，再到形成正式销售的产品，在这一过程中介绍了很多实用的知识和技能。虽然这些知识或技能单独拿到哪个细分领域中都显得有点简单，但如果想都了解它们并且知道怎么运用还是有难度的，而作者将帮助你做到这一点。

因为本书涉及的学科门类众多，加之译者水平精力都有限，虽然在保证翻译质量方面自觉已非常尽力，但还是有可能在翻译过程中引入错误。恳请大家在阅读过程中发现问题后不吝赐教。既可以在图灵社区提交勘误，也可以用email向我反馈（czhang@shnu.edu.cn）。非常感谢！

最后，肯定要感谢我的家人对我的支持。另外，要感谢图灵公司各位同仁的大力帮助，感谢武卫东、傅志红、朱巍、李鑫、周静和刘美英在本书引进、翻译和出版、营销等各个环节上的辛勤付出。感谢上海师范大学科技处、教务处和信息与机电工程学院相关领导和同事的帮助和支持。

张崇明

2014年9月于上海

致谢

首先，我们要感谢Wiley公司的Craig Smith，是他让我们相信，把我们知道的一些东西汇集成书是一件好事。也要感谢Wiley团队的其他成员和所有参与本书编辑的人，他们是：John Sleeva，Daniel Soltis，Alexandra Deschamps-Sonsino，Chuck Hutchinson和Aaron Crane。尽管我们对重写部分章节的额外工作有所抱怨，但这样做的结果是让本书变得更好了。

我们也想感谢利物浦DoES的每一位成员。在快到截稿时间的一段时间里，我们有时要分心写书，他们对此能一直保持容忍。特别要感谢DoES空间的其他几位组织者：John McKerrell、Andy Goodwin、Paul Freeman、Patrick Fenner、Steve Sparrow和Ross Jones。他们不得不在空间的运作方面承担更多的工作。

——艾德里安和哈基姆

我以前是从来不会太关注致谢部分的。作者会在这里感谢很多我并不认识的人。编写完一本书后，作为作者，我意识到自己得到了太多帮助和支持，因此对致谢部分有了新的认识。

我要感谢哈基姆同意和我一起编写此书。在一些我涉猎不深的领域，Andy Huntington、Lawrence Archard和Christopher Pett都慷慨的付出了时间，与我分享了他们的专业知识。

我需要感谢我的家人和朋友们，特别是Francis Irving、Andrew Dixon、Rob Stock、Neil Sowler、Kieran MacCourt、Jen Forakis、Colette MacCourt、Dan Lynch和Neil Morrin。他们给了我无尽的鼓励，使我在面对困难时能保持头脑清醒。

最后要感谢Bubblino，是它最先帮助我进入这一领域的。

——艾德里安

我往往至少会略读一下致谢部分，知道（在理论上）写书是一个巨大的协作项目，而对这一过程的亲身体验是令我惊叹的。我们几乎肯定

会忘记感谢某个人。如果这个人就是你，我们很乐意请你喝一杯你选中的饮料，以此作为补偿。

首先要感谢艾德里安能建议我们共同完成此书。这是一个有趣的项目。Dave Cross、Kieren Diment和Jodi Schneider给出了和出版有关的宝贵的早期建议。Georgina Voss为道德伦理一章的编写指明了正确的方向。Coen van Wyk校阅了初稿。Tomas Doran对于在线组件一章中有关异步Web编程的内容给出了有用的建议。Aaron Crane挺身而出，承担了技术编辑的工作。Kelvin Lawson告诉了我们实时操作系统的现状，而Kiel Gilleade和Jen Allanson在生理计算方面提供了深入的见解。我得到了Guy Dickinson一贯的热情支持。Jim Hughes使我能经常获得最新的有趣想法。还要感谢Electric Imp公司的Shaun Myhill，.:oomlout:.的Aaron，以及Antony McLoughlin、Ross Jones和Francis Fish。没有他们捐赠或出借的微控制器和书籍，编写此书是不可能的。我在mySociety的同事对我在定稿前的最后时刻休假写书的做法，也表现出极大的宽容。

有机会参加诸如开放物联网大会（OpenIoT）、IoT Howduino、TSB的IoT圆桌会议、BBC的PlayIoT非传统会议、MadLab的IoT研讨会、Internet World大会、LivLug和SMC Liverpool之类的活动（并且有机会在其中的一些活动中发言），促成了写这本书的想法。感谢Alex Deschamps-Sonsino、Ed Borden、Usman Haque、Adam Greenfield、Hannah Goraya、Nick O'Leary、Ben Ward、Thomas Amberg、Charalampos Doukas、Stefan Ferber、Chris Adams、Gavin Sparks、Peter Bihr、Andrew Back、Laura James、Russell Davies、Rob van Kranenburg、Erik van der Zee、Martin Spindler、Matt Biddulph、Fiddian Warman、Rachel Jones、Alistair MacDonald、Natasha Carolan、Mark Holmes、Pat Link、Cefn Hoile、Thom Shannon、Alistair Houghton、Chris Holgate、Alastair Somerville、Matthew Hughes、Jessi Baker、Aine McGuire和Aidan McGuire、Grey Povey、Hwa Young Jung、Andy Piper、Sue Black、Travis Hodges、Bob Ham、Neil Morrin、Dan Lynch、Francis Irving、Conrad Mason、Pete Thomas。此外，我在线上和线下与Javier Montaner、Paul Kinlan、Luke Petre和Paul Ede进行了很多讨论，并且从Rob Nightingale、Zarino Zappia、Aldo Calpini、David Jones、Kirsty Sparrow和Arto Nabito那里获得了有益的评论和建议。

早在2005年，我就开始和Greg McCarroll共事了。尽管在当时我没有意识到，但他的能联网能感知火车时刻的闹钟是我见过的第一个物联网装置。Greg是一位绅士，也是一位创新者。我不会忘记他。

还应该感谢Claire、Lolita、Rumplestiltskin、Sheri My、Wolfie和Helvetica。这是肯定的。

——哈基姆

引言

曾几何时，英特尔i486这样的计算机处理器和一部小汽车价格差不多。现如今，相似性能的处理芯片售价却和一根巧克力棒差不多。

处理器变得如此廉价，它的应用场合就更广了。除了可以在商用工作站和家用PC上使用，电话、电表、床头灯或泰迪熊等也都可以内置处理器，这在以前可是负担不起的。我们可以使物品智能化，让它们能够思考说话。按专家们的说法，这就是“物理计算”（physical computing）、“普适计算”（ubiquitous computing或ubicomputing）或“物联网”（the Internet of Things）。不管使用上述哪一个术语，我们实际讨论的事情是一样的，即制作奇妙、有魔力的物品。

在本书中，我们将聚焦于可以嵌入到物品中的计算机硬件（诸如Arduino之类的微控制器），一步步引领读者经历从原型系统设计与制作到成品制造与销售的全过程。我们将介绍相关的软硬件开发平台，讨论能使你的产品吸引眼球和讨巧的设计理念，向你展示怎样把单个原型提升为可以批量生产的成品。

本书的局限

首先，本书不是针对任何特定类型微控制器的指南。虽然我们将介绍 **Arduino**、树莓派（**Raspberry Pi**）和其他可选用的微控制器，但和本书介绍的其他内容相比，这种详细的技术信息将不可避免地更快过时。因此我们更愿意展示的是评估和选择平台的标准。

其次，本书也不是针对某个很酷项目的实战指南。我们将对一些有开创性的物联网装置进行综述，主要关注其通用设计原理，希望借此来鼓励读者制作新颖、美丽、实用和有魔力的物联网装置。

最后，本书不是介绍未来物联网商业基础设施的学术专著，诸如 **6LoWPAN**和新兴的**M2M**标准等都没有专门介绍。我们对怎样设计、制造和销售面向消费者的、能让人欣喜的物联网装置更有兴趣。

本书的目标读者

作为一本技术出版物，我们希望本书能对开始从事物联网产品设计的软件工程师、**Web**开发人员、产品设计师和电子工程师有所帮助。实际上，本书内容涉及微控制器、电子技术、嵌入式编程和**Web**编程接口（**API**）等诸多有趣的技术话题。

除此之外，包括企业家、创客群体（设计师、艺术家、手工艺人和业余爱好者）、学术界人士和教育工作者在内的任何人，只要想对物联网这一激动人心的新兴技术有一个概貌性的了解，都是本书的目标读者。即便你没有多少或完全没有**IT**方面的技术背景，也可以读懂本书的大部分内容。我们谈论的话题超出了纯粹的数字技术范畴，还涉及了设计、道德伦理和商业。

怎样使用本书

前面已经说过，本书不属于“指南”类图书，因此阅读本书时，你不需要准备什么特别的工具。偏重技术的几章会针对微控制器硬件或Web开发框架给出一些建议，你可以在阅读本书的同时做更进一步的研究。

本书的内容安排，从前到后依次为原理介绍、原型系统设计与制作、制造和商业方面的考虑，因此你当然可以从头到尾地阅读。不过，你也许更愿意采用其他方式阅读本书。根据自己的知识背景和关注点，你也许会有选择性地阅读某些感兴趣的章节，而暂时跳过其他章节。

第一部分是“原型阶段”。本部分首先介绍了物联网的一些基本知识，然后进入实践环节，介绍了原型项目的创建。

我们建议读者从头开始看。第1章“物联网概览”，介绍了什么是物联网，为什么它会在现在出现。第2章“联网装置的设计原则”，介绍了设计面向消费者的物联网装置时，需要遵循的一些设计原则。

第3章“因特网原理”，采用容易理解的方式，介绍了因特网的基本原理。当你构建物联网装置并考虑如何联网时，这些内容会对你有所帮助。如果你有因特网方面的背景，熟悉Web相关的各种网络协议，当然可以选择跳过这一章。

如果你打算自己制作物联网装置，那你应该对下面几章最感兴趣了。首先，第4章“原型设计与制作概述”有助于你更好地理解这个领域，对各种技术选择有一个总体的把握，值得一读。第5章“嵌入式装置的原型开发”，面向负责物联网装置制作的工程师、创客和技术人员，把第4章介绍的基本原理应用到了特定的平台。第6章“原型系统的结构设计”，讨论了怎样为你的原型系统制作机械构件。第7章“原型系统在线组件的设计”，介绍了作为Web接口的在线组件的构建。

虽然很多读者都有编程方面的背景，但对于通常为物联网装置提供处理能力的微处理器系统，为其编写代码有一定的挑战性。第8章“嵌入

式编程技术”，介绍了从编程实践中获得的一些经验知识，对在项目中涉及复杂计算需求的创客们来说，这些内容是会有帮助的。

第二部分是“产品阶段”。这部分将不再谈论原型系统的设计和制作，而是关注在原型项目转化为产品的过程中发生的事情。

如果你是一个企业家，希望能从物联网项目中盈利，那么你会关注第9章“商业模式”。商业模式不是简单的卖设备，而是具有更深层次的含义。当然，如果你正在筹划销售物联网设备，那么进入制造阶段后你将面临很多全新的问题，诸如制作PCB、采购原材料和获得认证等。在第10章“生产制造阶段”中将会讨论这些内容。

最后，技术虽然改变世界，但未必总是朝着好的方向改变。我们在本书的开始部分介绍了物联网装置的设计原则。第11章“道德伦理”告诉我们，为了避免这些有魔力的物品被不当利用，恪守伦理和道德准则是必要的。

你可以访问book.roomofthings.com 或者关注Twitter账号 @aBookOfThings，了解更多关于本书和两位作者的信息。

写作手记

艾德里安 (Adrian)

人们常说旅行能开拓视野，实际上旅行也有助于写作。本书的部分内容是在飞机上写的，但更多的内容是在无数次的火车旅途中完成，乘车区间主要是在利物浦和伦敦之间，也包括英国境内的其他地方和意大利北部。昔日的交通系统也对写作有所帮助。在本书创作初期，我曾长时间在纽约逗留，发现纽约高铁公园是一个极好的写作场所。

本书的其余部分是在利物浦及其周边地区完成的，包括利物浦的DoES创客空间、大教堂附近的我的公寓、波德大街上的咖啡馆和宏伟的中心图书馆三楼。天气条件允许时，我甚至去皮尔希德码头能俯视默西河的地方写作。

码字工作是在一台索尼Vaio笔记本上完成的，在Ubuntu系统下用Vim编辑器以Markdown语法格式完成了本书主要的内容，之后切换到该电

脑的Windows系统，用Word进行了编辑。

哈基姆 (Hakim)

写作本书之前，艾德里安和我就物联网的总体情况和本书内容，有过几次长时间、内容广泛的讨论。之前我并不了解物联网，通过问一些愚蠢的问题和质疑各种假想，这些讨论帮助我了解了物联网。这些讨论也有助于我们对本书的内容形成一致的意见。我们共同起草了一章（最终形成了第4章），然后根据各自的兴趣和知识结构，对其余各章的撰写进行了均等的分工。我们始终坚持在提交各章之前，互相审阅对方写的内容，这有助于维护全书内容的一致性。

我用Vim撰写文稿，用Pandoc对Markdown格式的文稿进行转换，然后在LibreOffice中进行编辑。最初用的是一台老旧的ThinkPad电脑，后来坏掉了，就改用MacBook Pro了。Dropbox网盘被用来即时地共享最新版本的文稿，非常好用。我们撰写博客文章也是用Markdown语法，并且使用Jekyll生成网页。

第一部分 开发阶段

- 第 1 章 物联网概览
- 第 2 章 联网装置的设计原则
- 第 3 章 因特网原理
- 第 4 章 原型设计与制作概述
- 第 5 章 嵌入式装置的原型开发
- 第 6 章 原型系统的结构设计与制作
- 第 7 章 原型系统在线组件的设计
- 第 8 章 嵌入式编程技术

第 1 章 物联网概览

我们首先需要解释一下什么是物联网。虽然本书中提到的概念都相对简明易懂，但人们对物联网这个技术术语的理解却不尽相同，很多相关的表述也并不容易理解。因此，在本章中我们并不急于马上给出解释，而是先从不同视角了解一下物联网。

“物联网”这个术语是什么含义？它和以前说的“普适计算”又有着什么样的联系？那些对技术发展历史感兴趣的人，想知道物联网在物理世界中处在一个什么位置，为什么现在开始热议它。对于那些善于通过隐喻理解问题的人，我们需要借用“有魔力的物品”这一概念。虽然用这个词来描述技术已沿用千年，但用它来解释物联网的概念尤其有效。而对于更多的善于通过实例理解问题的务实读者而言，我们则需要给出一些令人兴奋的项目，来展示这一激动人心的领域的各个方面。下面，让我们采用实例讲解的方式，设计一些虚构的情节开始介绍吧。

1.1 物联网的应用范例

闹铃响了。你睡眠惺忪，发现醒来时间比平时晚了5分钟。这是因为闹钟已在线核对过火车时刻表，发现你要乘坐的火车晚点了，于是让你多睡一会儿。（参见<http://makezine.com/magazine/make-11/my-train-schedule-alarm-clock/>。）

在你的厨房，有一个灯通过闪烁提醒你服药时间到了。如果你忘了服药，药瓶的盖子会自动联网向你的医生发送电子邮件，让她知道你没按时服药。（参见<http://www.vitality.net/glowcaps.html>。）

你正打算出门，眼角的余光突然注意到一丝光亮，原来是雨伞的伞柄在发光。这意味着它已经查看了BBC的天气预报，预报说今天有雨。你叹了口气，把伞带上。（参见<http://www.materious.com/#/projects/forecast/>。）

你在去地铁站的路上，路过一个公交站台。你注意到硕大的LCD显示屏正显示23路公交车即将到达。当你走到下一个路口时，这辆车到了。公交公司最初安装这些显示屏的时候，它们只能用来显示预定的时刻表。但现在每辆公交车都采用了GPS追踪定位，并连接到了公交公司的在线服务平台上，不断地更新着自己的位置信息。目前，很多交通运输组织都已经实现了这个功能，你可以通过以下网址访问伦敦交通局，来查看一些关于公交信息指示牌的有用信息。（参见<http://www.tfl.gov.uk/corporate/projectsandschemes/11560.aspx>。）

到达地铁站时，你的手机会自动通过定位服务（例如Foursquare）查看你所在的位置。同时，你家中的壁炉架上，一个带转盘的装饰物会注意到你位置的改变，转盘开始旋转，转盘的指针指向“途中”的标记。而当你安全到达工作地点时，指针会转到“工作”的标记，这样你的家人就知道你已经安全到达工作地点了。（参见<http://wheredial.com>。）

你午休时，运动鞋中的计步器和手腕上戴的心脏监听器将跟踪你在街区周边的跑步情况。腕带上的大号显示屏也方便你低头查看跑步速度和你所消耗的热量。之后所有这些数据都将自动上传到你的运动跟踪

网站，该网站同时也整合了你的在线超市购物账号，这样可以方便你将运动所消耗的能量与你所摄入的能量做比较。（参见 <http://nikeplus.nike.com/plus/>。）

正如你在上述链接中所看到的那样，这里提到的每一个产品都可以利用现今的技术实现。每个产品都有了原型，很多产品则已被做成工艺品或大众消费产品。

1.2 物联网的概念

我们已经看了几个物联网的范例，那么它们之间的共同点在哪里？我们又为什么要把它们叫做物联网呢？可以看到，上面的所有实例都使用因特网发送、接收或交换信息。在每个实例中，连接到因特网的装置不是计算机、平板电脑或移动电话，而是某种日常生活中的实物，或者说物品。这些物品都针对一定用途进行了特意的设计：雨伞需要有一个可以伸缩的伞面和一个可以支撑伞面的手柄；公交车的显示屏需要让乘客（包括年长和有视力障碍的乘客）觉得清晰易读，且需要能扛住恶劣天气，还要考虑到人为破坏的风险。运动手环则要方便跑步时佩戴，要有一个足够大、足够亮的屏幕，便于佩戴者在跑动中查看，还要能经受冷、热、出汗和下雨的考验。

物联网应用的很多范例常常能够用通用计算机实现。尽管我们不会扛着台式电脑到处跑，但很多人确实总是随身携带笔记本电脑或平板电脑。特别是现在几乎每个国家的大多数人都不会随身携带手机。这些手机往往是有足够计算处理能力的智能手机，计算机能做的事情，它们也能做。让我们看一下用手机代替智能物品做类似事情的效果如何。

乍看起来，用智能手机的网页浏览器查阅公交运营商的时刻表也能实现相同的功能。注意这里的措辞“乍看”——乘客到达公交站台的时候，只需扫一眼显示屏，就能知道下一辆车什么时候到，而用智能手机做这个事情就麻烦多了。你要有一部智能手机并且能承受数据流量费用（如果你去国外旅游，这个费用会相当高），你需要从口袋或包中拿出手机，解开屏锁，访问正确的网址（不管你是直接键入网址，还是使用二维码，这可能都是最慢、最复杂的一个步骤），再从小小的屏幕上查看数据。在这期间，你不能时刻留意到站的车辆，甚至有可能错过你要乘的车。

你也可以用智能手机上的应用程序跟踪你的跑步状况，很多人也的确这样做了。智能手机有GPS功能，具备很多有用的传感器、数据处理能力、因特网连接和不错的显示屏，但事实证明，这样的手机在跑步时并不方便携带，你要担心它会掉落或者被弄湿。当然，有多种携带手机的方式可供选择，包括使用腰包和臂带。使用臂带理论上使你能够在跑步时查看手机，但实际上当你的身体上下晃动时，很难看清手

机屏幕显示的细节内容。为了解决这一难题，诸如RunKeeper这样的手机应用程序提供了实用的定时语音播报概要的功能

（www.runkeeper.com）。总之，用手机跟踪跑步过程是完全可以的，多数跑步者会认为用手机记录跑步数据是一个充分、舒服自在和有趣的方式。然而，其他人会更愿意选择像手表和腕带这样的可穿戴装置，因为它们在运动状态下方便读数，又可以在下雨时穿戴，并且还可以连接心脏监视器等外部设备。

当然，手机（甚至平板电脑或笔记本电脑）还不够大，没有足够的防水性能，不可以直接当雨伞用。但你可以出门之前，把智能手机和普通的雨伞关联使用，通过查看手机应用程序来了解是否可能下雨。但这与你出门前路过伞架时，眼角的余光就能注意到来自智能雨伞的平静而微弱的光线，并对这个环境信息进行下意识处理不同，使用手机应用需要完成若干个操作步骤才能获得类似的信息。如果你养成了用手机查询信息的习惯，这样做也未尝不可。智能雨伞并不需要具备比手机应用程序更强的功能，它只是把相同的智能融入到了环境中，这样你的日常生活就无需改变什么。

综上所述，物联网的理念是：不要求你在日常生活中拥有少数几台功能强大的计算设备（笔记本电脑、平板电脑、手机和音乐播放器），但可以有很多性能一般的智能装置（雨伞、手环、镜子、冰箱和鞋）。早于“物联网”出现的技术术语“普适计算”（ubiquitous computing，它们有个难看的组合词ubicomp），表达了差不多的理念。“普适”一词也反映出，具有计算功能的物品会非常多。因特网现在已经成为数据传播的主要渠道，很难想象还会有不具备持续的宽带接入能力的PC机，年轻的读者甚至可能从来没见过不能上网的电脑。技术专家和专栏作者拉塞尔·戴维斯在伦敦举行的2012年度开放物联网大会（Open Internet of Things Assembly）上开玩笑说：

“我不理解，为什么以前的泰迪熊没有WiFi连接能力。没有WiFi的泰迪熊奄奄一息，没个熊样。”

——<http://storify.com/PepeBorras/opent-iot-assembly>

普适计算的定义涵盖了更广泛的领域。例如，Glade空气清新器在检测到房间内有人活动时，会释放香味，这也算是普适计算的范畴了。也

就是说，符合普适计算定义的设备包括一个智能的可编程的计算机处理器，该处理器能被现实世界中的传感器所驱动，同时也能驱动现实世界中的输出装置，并且这些处理器、传感器和输出装置都完全嵌入到了日常生活中的物品里。普适计算和物联网两者的不同之处仅仅在于，现如今大多数真正有趣的有计算能力的物品都具备因特网的接入能力。

把物品接入因特网意味着什么呢？很显然，在椅子上安装一个以太网接口和在缝纫机上安装一个3G调制解调器，并不能把它们立刻变为拥有神秘属性的物品。需要某种信息流的存在，把物品的典型特征和因特网内的数据和数据处理系统联系起来。

此类物品确实就存在于真实环境中，存在于家庭、工作场所和汽车中，或者会被穿戴在身上。这意味着它可以从你周围的环境中获得输入信号，把信号转换为数据后，再发送到因特网上进行收集和处理。所以，你的座椅可以知道你就座的频率和时长，并收集这些信息。同样，缝纫机可以报告缝纫线的剩余情况和已经缝制的针数。在后续章节中，会有很多和“传感器”有关的内容。

此类物品的真实存在也意味着它可以使用“执行器”，在真实环境中产生输出动作。其中一些输出动作可以被因特网内收集和处理的数据所触发。因此，你的座椅可以通过震动通知你收到了新的电子邮件。

我们注意到，在上述所有范例中，物品的外形设计和它的基本功能相符。例如，椅子是用来坐的，缝纫机是用来缝衣服的，等等。虽然此类物品也具备因特网接入能力和通用计算能力，但这并不意味着它的外形需要改变。（也许有人会说，现如今的智能手机和平板电脑在外形上的设计，其实是冲着通用计算机的功能去优化的，并不是为通话功能考虑的。面对如此众多的触屏手机，人们可能会问，它们在设计上是否考虑了易于持握、不容易掉落和日常使用中的撞击等因素。）

我们可以用下面这个非常简化（但也有点过度简化）的等式总结一下物联网的构成元素，见图1-1。



图 1-1 物联网的构成

1.3 物联网出现的技术背景

前面在定义物联网概念时，把它和稍早出现的普适计算做了对比。我们可以继续把它和比尔·盖茨在1977年提出的著名愿景“让每个家庭以及每个桌面上都有一台计算

机”（http://danbricklin.com/log/billg_entwof.htm）做比较，还可以把它和更早出现的观念“计算机是一种非常昂贵，专供大学、有前瞻性视野的全球化大公司和军方使用的机器”做比较。总之，为了更清楚地理解物联网和它适用的范围，花一点时间，从历史发展的角度审视一下物联网是值得的。

最初，技术的最大驱动力是人们对食物、水、温暖、安全和健康等的基本需求。打猎、觅食、生火、建屋、筑堡和医疗都来源于这些需求。后来，因为这些技术所需的资源并不总是分布在人们期望的地点和时间，让人们和他们的财产、牲畜和其他资源能够移动的技术就应运而生。贸易就是随着货物被从丰富和便宜的地方运到该类货物稀少和价格高的地方发展起来的。贮藏可以看作是货物在时间上的移动，例如，在丰收季节，食物丰富且价格便宜，就将食物储存到价格高企的冬季。

语言的产生方便了技术的交流，从此信息也成为了一种重要的资源。旅行者可以在交换货物和服务的同时传递消息，口述的传统风俗使得信息可以在时间上和空间上传播。文字的发明使得信息的交流更加重要，从古代的哲学家和诗人到当代的作者，都可以通过自己的文字和自传让自己的生命得以留存。从书面文字，到电报、无线电广播、电视，再到数字化的信息，有越来越多的技术支持信息的流动和使用信息做有趣的事情。

但前面所述的人们的各种基本需求并没有因为信息时代的到来而消失，将来也不会。人们现在仍然需要吃饭喝水，需要光和热，需要爱情和友情。人们仍然需要椅子、衣服和鞋子，需要各种交通和交流方式以及娱乐方式。这些事物的外在形式和具体细节将来会有所变化，但它们所解决的各种需求并不会变。

随着技术的发展，出现了新种类的物件。电话、收音机、电视、计算机和智能手机都是电子时代的产物。和大多数新技术一样，这些装置刚出现的时候很贵，之后其价格会一路下降。需求驱动了价格的下降，研究促进了优化和小型化。最终，原本需要用一个专门的装置实现的功能，现在可以作为另一个装置的附属功能实现，这种做法的可能性和可行性现在都没有问题。电视屏幕最初在起居室里会占据很大空间，现如今，平板电视面板不仅更省空间，而且这项技术已经变得无处不在。能够显示电视节目的高清显示器可以被嵌入到门框中或厨房设备中，而小屏幕显示器则出现在音乐播放器和手机中。

和计算机的情况类似，嵌入到装置中的通用芯片的生产成本也已经变得非常低廉，以至于你的洗衣机中可内置一台搭载Linux的计算机，超市的收银机可在Windows系统中运行，你的视频播放器可能运行的是苹果公司OS X系统的某个版本。但就像我们在前面已经间接表明的，仅仅有计算能力不是构成物联网的充分的前提条件，我们要看这个计算能力是不是一方面与传感器和执行器这样的与现实世界互动的实物相关联，另一方面又与因特网相接。能与网络服务或其他数据消费者快速共享和处理信息是物联网的重要标志。

我们举个例子，看一下汽车中的计算机。现代的汽车使用数量众多的传感器，对油量、胎压和发动机的内部进行检测，用来确定汽车的运行状态。除了做诊断测试，当处理器发现诸如车轮抱死或转速失控等情况时，由计算机控制的刹车装置可以协助司机处理故障。虽然涉及的数据处理和分析可能非常复杂，但都是本地信息的处理，终究是由汽车制造商编程设定的。也许你的汽车会使用GPS确定你的位置，这就会产生来自外部的数据（不一定与因特网相关）。出于保险和防盗的目的，高端的车型可以向跟踪服务商报告位置信息。这种情况下，汽车搭载的计算设备不仅可以被动消费数据，也可以和外部服务交换信息。当汽车上搭载的计算机和因特网相连时（定时地或持续地相连），就有可能提供根据实时路况调整行车路线这样的服务。虽然你的车载GPS设备可能已经提供了这些行车路线数据，但现在你可以根据附近其他联网司机的行为，对数据进行聚合，采用“社交化路线规划”的方式，实时地调整行车路线。把汽车内部的数据接入因特网，并且对这些数据进行处理和分析，与其他数据聚合和混合之后，将会开启各种可能性。这些可能性不仅仅是各个相关领域中业已存在的可能性，也包括我们还没能想到的新的可能性。因此，当你把处理器嵌入

到物品或家电中时，这是一个真正的改变；当你再把处理器接入因特网时，这又是一个真正的改变。我们有必要了解一下，后面这个改变为什么会在现在这个时代出现。

20世纪80年代末，当因特网走出学术圈和军事领域，第一个商业化的因特网服务提供商（ISP）开始运作时，早期的因特网商业用户可能使用搭载英特尔i486处理器的计算机上网。这个i486处理器当时差不多要值1500英镑，是一部小汽车的价钱。现如今，相同计算能力的处理器芯片大约只值0.5英镑，只有一根巧克力棒的价钱。快速增加的处理能力和伴随而来的成本下降不是什么新发现，其实就是广为人知的摩尔定律（英特尔公司的联合创始人提出的经验规则，认为一枚芯片上可以容纳的晶体管数量每18个月会增加一倍）。

上述芯片价格的差别不仅仅只是一个程度的问题，它既是量变也是质变。这是一个长尾现象，我们已经到达了一个性价比的最佳位置。这意味着，把物品接入因特网的成本已经降到了一个很低的程度，给物品添加网络或计算能力近似于选择哪种材料和漆料，例如是否选用稍贵一点儿的木饰板。无论哪种选择都会增加一点产品的成本，但对消费者而言，增加的价值可不只是一点点儿。当接入因特网需要花费数千磅时，我们不会考虑给物品增加联网能力，但现在这样做的成本只有几十便士，接入因特网就成为了一种可选的功能。

可见，计算能力的价格已经下降到了一个负担得起的程度，这还不是故事的全部。从洗衣机到汽车，电子产品的制造商们开始把通用的计算机处理器嵌入到他们的产品中，因为他们发现，很多情况下，这样做比设计和使用专用的芯片划算。这些通用平台拥有的大量编程和调试资源，对爱好者们和原型系统设计市场很有吸引力，进而促进了微处理器市场的快速增长（我们将在第4章和第5章中介绍相关内容）。

和过去相比，接入因特网也变得更便宜更方便了。过去我们上网是使用拨号连接，既贵又慢。现在在英国，76%的成人使用宽带上网，能持续保持网络连接。有线形式的以太网提供了即插即用的联网体验。现在多数的家用路由器也提供WiFi功能，免去了到处拉网线的麻烦。

位置固定、有因特网接入能力的计算机用来工作或学习是很方便的，但它们常常被家庭中的男性和年轻成员所占据，主要用来浏览网页或

玩游戏。既然现在全家人都可以舒服地在客厅沙发或各自的房间里上网，人们就对使用网络更有信心，倾向于更多地使用网络了。

希望读者能原谅前面的泛泛而谈。如图1-2所示，在英国，从2002年开始，16~24岁年龄组男女使用计算机的比例几乎相同。对于55~74岁年龄组，男女使用计算机的比例尽管持续在增加，但在2010年前后到达临界点之前，一直有一个明显的差别

(http://w3.unece.org/pxweb/database/STAT/30-GE/09-Science_ICT/)。我们假设，这种改变至少可以部分归因于处理能力和联网正变得更便宜、更方便且能随处获取。并非完全巧合，我们认为物联网的兴起也和这些因素有关。

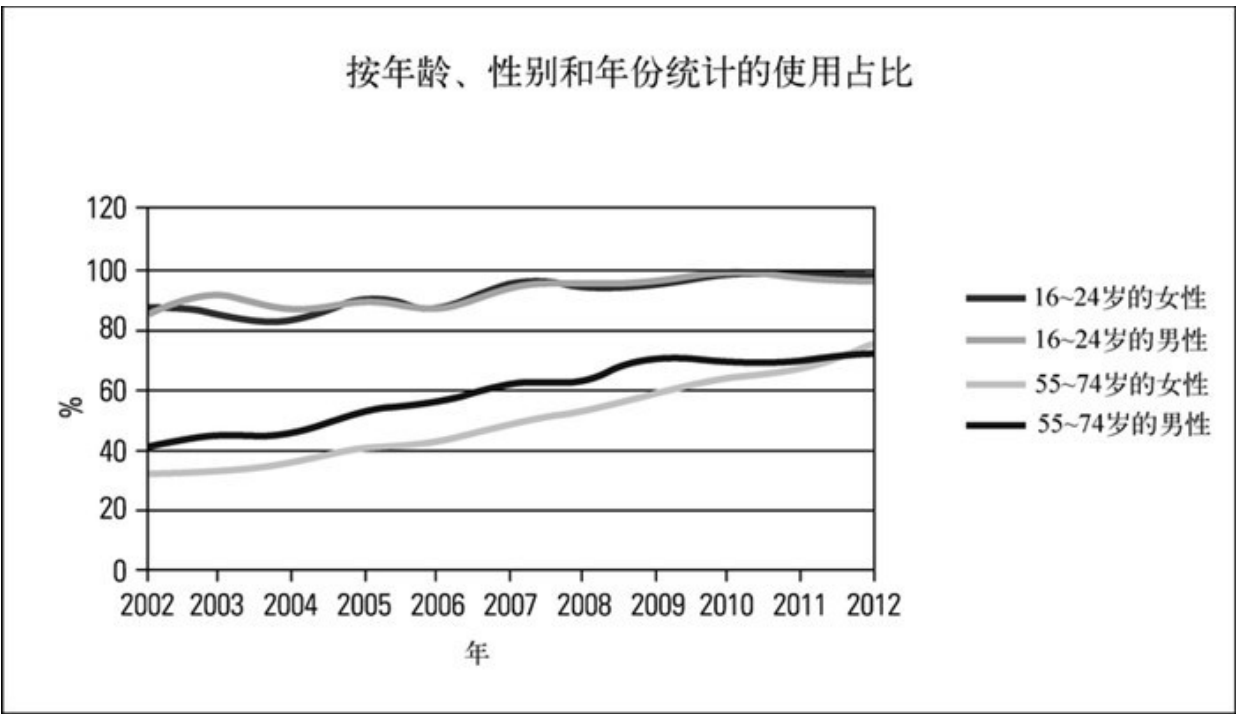


图 1-2 UNECE关于性别和计算机使用率的统计

联合国欧洲经济局（UNECE）关于性别和计算机使用率的统计

在没有固定网络连接可用的环境下，移动网络被广泛使用。因为联网的需求非常巨大，甚至诸如空白频谱网络这样不太成熟的解决方案也已开始使用。空白频谱网络利用已淘汰不用的模拟电视网络所占用的频谱空间，弥补了联网频谱资源供给不足的缺口。

另一个起作用的因素是因特网平台的日益成熟。早期的Web应用只能通过浏览器使用。Web 2.0作为一项前瞻性的技术，除了带来“Web富应用”外，也使得利用应用编程接口（API）进行编程变得更为流行了。除了用户，其他程序也都可以通过使用API与网络服务交互，使用网络服务提供的功能。这就形成了一个完备的生态系统，其他网站可以通过综合多个网络服务的方式构建适合手机应用程序和联网装置使用的新Web应用。

随着因特网服务的成熟，用来构建和调整扩展这些服务的工具也日益完善。诸如Python/Django和Ruby on Rails等Web服务框架都允许快速构建在线组件的原型。与此类似，诸如亚马逊Web服务（AWS）这样的云服务，就是一种方便调整扩展的解决方案，即随着服务受欢迎程度的增加，可以在使用的同时方便地进行扩展。在第7章“原型系统在线组件的设计”中，我们将介绍面向物联网应用的Web编程。

1.4 有魔力的物品

亚瑟·查理斯·克拉克（Arthur C. Clarke）在著名的“预测三法则”中指出：

任何足够先进的技术其实别无异致。

——http://en.wikipedia.org/wiki/Clarke's_three_laws

我们已经看到技术通过怎样的演进满足了我们的需求和愿望。魔法的演化发展在很大程度上也有着相似的功用。毕竟，民间故事和童话故事中的物品常常是为达成愿望幻想出来的，用来满足内心的渴望：我要是能有足够多的食物多好，要是我妈妈能康复多好，我要是能和远方的朋友交谈多好，我要是能回家多好，我要是能不用每天都超负荷工作并且又能赚足够的钱养家多好。文学和人类学领域的学者对童话故事进行了长期的研究，分析了故事人物、故事情节和故事中的物品，为的是从故事本身及其内涵的基本规律中获得启示。例如，形式主义学者弗拉基米尔·普洛普（Vladimir Propp）对其祖国俄罗斯的民间故事进行了分类，从故事情节归纳出了故事的31种功能，包括“违背禁令”、“罪行”、“领受神力”、“艰巨任务”，等等。

最近，戴维·罗斯（David Rose）从一名硅谷企业家和技术专家的视角，在伯克利TEDx活动中讨论了有魔力的物品

（<http://tedxtalks.ted.com/video/TEDxBerkeley-David-Rose-Enchant>），并且把来源于童话故事和幻想文学的各种物品按其应用像技术物品一样做了分类。

- **保护类物品**：正如魔剑和头盔可以保护童话故事的主人公不被敌人伤害一样，历史上的科技发展在很大程度上都是受获取军事优势这一需求驱动的，其目的是为了确保安全或征服对手。
- **健康类物品**：童话故事中的很多探索活动是在寻找生命药剂的某个成分，健康也是对医学、药理学与外科手术、理疗、饮食等各种科学分支进行研究的驱动力。

- **全知全能类物品**：白雪公主邪恶的继母会问魔镜“魔镜魔镜告诉我，谁是世界上最漂亮的人”，而人们常常通过在智能手机上查询维基百科的方式解决朋友之间针对某个事实的争论。
- **人际联系类物品**：当所爱之人在很遥远的地方时，相互联络是一种急迫、令人心焦的需求。芬兰史诗中的英雄人物勒明盖宁（Lemminkäinen）的家人看到他留在壁炉台上的魔梳开始流血，就知道他受伤了。同样，邮政服务、电话和社交网络也可以使我们和家人、朋友保持联系。
- **轻松移动类物品**：因为渴望能不费力气地到处移动，古代的故事作者们创造了神行靴、飞毯和瞬间转移技能，而我们则通过技术的变革发明了汽车、火车、自行车和飞机。
- **创意表达类物品**：童话故事通过使用魔法画笔、魔笛和魔琴来满足创意表达的需求。在现实世界中，从使用木炭绘画，到利用颜料绘画，再到使用计算机绘制图形，或者从鼓到小提琴，再到电子合成器，我们总是通过技术手段实现创新性的表达方式。

所以，技术总是会和魔法发生联系的。对物联网而言，也是差不多的情况。不仅如此，很多有魔力的物品还有一个关键元素：除了实际的魔力，它们还有名字和个性，这意味着它们拥有实现物品的基本功能之外的额外智慧。此类物品的例子很多，从芬兰史诗中名为Sampo的神磨、英国亚瑟王的神剑，到托尔金（Tolkien）和莫考克

（Moorcock）小说中具有邪恶智慧的至尊魔戒和“兴风者”魔剑，每一种物品都有各自的个性和品行。正如这些具有魔力的磨、剑和戒指拥有基本功能之外的能力，联网装置具有的处理和通信能力也使其不再是一个普通的灯、雨伞和泡泡机。

1.5 物联网的创造者

虽然本书会介绍各种理论方面的内容，但我们主要还是对实际**设计**和**制作**物联网装置的实践过程感兴趣。物联网领域的思想领袖和企业家亚历山德拉·德尚-桑西诺（Alexandra Deschamps-Sonsino）在维多利亚和阿尔伯特博物馆举办的“造物的力量”（Power of Making）研讨会上指出，**设计**和**制作**这两个词对不同人有不同的含义。图1-3描绘了她最初用来表示“造物”含义的尝试。

图1-3所列的各个学科之间有很多交叉点。艺术家可以和设计师在装置艺术方面进行协作，也可以和传统手工艺人在版画复制方面进行合作。设计师和工程师通过紧密协作制造工业产品。爱好者中的“黑客”们（这里泛指喜欢捣鼓小发明的人和非专业的工程师）天生就是一个多样化的群体，对各种技术和艺术感兴趣并具备技能。上图所列学科未必详尽，你也许对建筑师的角色被遗漏心存疑惑，这是因为建筑学科涵盖了上图中工程师、设计师和手工艺人的角色。

鉴于德尚-桑西诺物联网创新者的身份，图1-3中一个更明显的“疏漏”是没有“物联网构建者”的角色。当然这不是真的偶然的疏漏，而是因为物联网涵盖了图中所有学科：黑客可以捣鼓物联网装置的原型；软件开发者可以编写在线组件；设计师可以把丑陋的原型转变为漂亮的物品，而完成这一转变可能需要借助手工艺者的技能；工程师可以解决困难的技术挑战，特别是在从原型到扩大生产的过程中；最后，正如我们将在第2章中看到的，物联网应该是“漂亮的物联网”，就和采用手工方式和工程化方式设计制作的各种物品一样，也是或可能是出自艺术家之手。

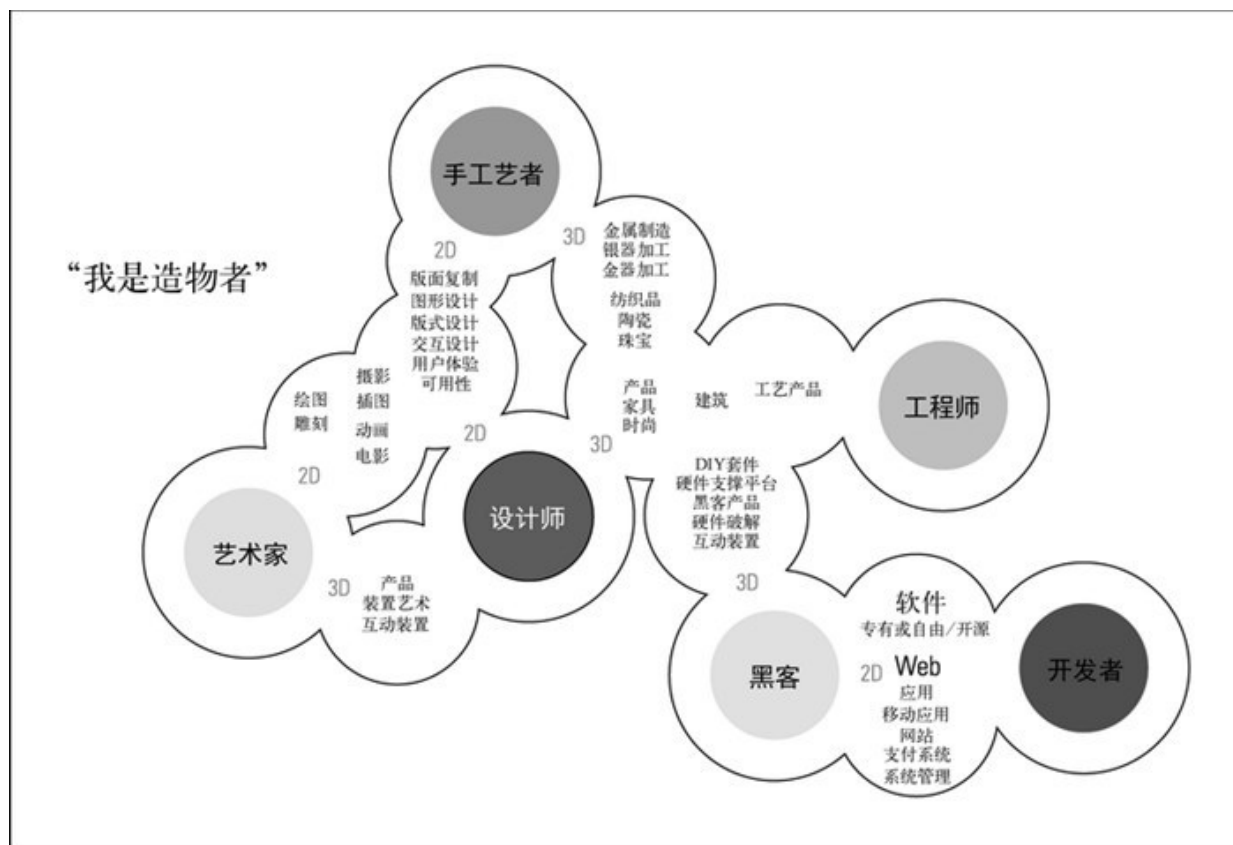


图 1-3 “我是造物者”

当然，如果对所有这些学科都能驾轻就熟，确实有助于创造出真正成功的产品。但现在很少有这类文艺复兴时期的全能高手了。如果你只能承担图1-3中的一个或几个角色，你仍然可以通过学习足够多的其他角色的技能来实现自己的创新想法。获得额外的技能，实现项目从原型到产品的转变，是值得做的事情。我们将在第9章和第10章介绍这些内容。我们想强调的最重要的一点是，不管你这个创新者的兴趣具体是什么，你都有足够的资格进入物联网这个激动人心的领域！

1.6 小结

在本章的开始部分，我们介绍了物联网的一些实际应用范例。无论是第一部分，从创建初始原型的视角做的介绍，还是第二部分，为了实现商业化，制造和配销它们所额外讲到的内容，很多类似项目的介绍都将贯穿全书。物联网的典型特征是把物品、嵌入在物品中的计算机，以及因特网上的通信和代码结合在了一起。在与原型系统的设计和制作相关的章节，以及与生产制造相关的章节，我们都会聚焦于这三个基本要素。

我们把物联网装置和有魔力的物件做了类比，这种类比从下一章开始也将贯穿本书。下一章是介绍设计原则的，我们希望这些原则有助于设计出优雅、好用、有趣和让用户感觉开心的装置。创造一个让人开心的有魔力的物品看似一个有难度的任务，正如我们在上一小节看到的，制作一个物联网装置所需要的专业技能是非常庞杂的。然而，这也意味着制作此类联网装置的竞争环境非常公平。不管你拥有的技能和兴趣是什么，你和任何其他刚开始尝试做这个事情的人处在相同位置。如果你想进入物联网这个激动人心的世界，现在就是最好的时机。

第2章 联网装置的设计原则

本书的原书名包含**设计**两字，这是不是意味着我们应该认为：在构建联网装置的过程中，设计起着十分重要的作用？

某些应用乍一看，你会觉得设计没什么重要的。谁会介意工厂里生产线上放置传感器的盒子的外观？功能决定形式，对吧？

和准备放在壁炉架上的物品相比，一般的联网装置的结构设计也许没那么重要。但其他功能性方面的设计却不能草草应付了事，装置的结构和它的控制系统、和工厂中的其他设备之间怎样相互作用，都是应该要仔细考虑清楚的。

你也许觉得设计仅仅是关注物品的形状和外观，做一些装饰性的事情，让物品看起来赏心悦目。如果你未曾花太多时间和设计师们交流，有这种想法可以理解。但实际情况是，设计所涉及的领域比你想象的要广泛得多。

工业设计（也称为**产品设计**）除了关心物品的形状和装饰，也涵盖了一些功能性的方面。例如，搞清楚产品应该怎样构造，确保操纵装置易懂好用，等等。

物品的用户接口（屏幕上的控件或传统的按钮和开关）也是体验设计学科感兴趣的内容。该学科从最终用户的视角观察设计，寻求建立符合用户需求的最佳解决方案。显然“最佳”是一个主观的目标，是令设备尽可能地有趣还是让设备尽可能有效地被使用，这要看设计师及其团队优先考虑的目标是什么了。

数字服务的兴起，特别是那些利用因特网以及由此产生的网络效应的服务，要求设计专家们能对整个系统的设计有一个更开阔的视角。服务设计是从整体上以最广阔的视野研究服务，而交互设计也研究系统的不同部分是怎样互相关联的，特别是用户在交互过程中扮演什么角色。

上述设计相关的各个学科之间没有明确的界限。我们认为这些学科的设计师们都会同意，设计可不仅仅是对产品外观的美化装饰。

本章将介绍几个在设计物联网系统时可以应用的非常重要的原则，并且讨论几个有助于探索问题域和最终形成好产品的技术。这些技术未必适用于所有的情况，但它们可以在你着手做设计时，提供一些有用的经验规则。

2.1 环境计算和宁静技术

物联网源自于20世纪90年代马克·维瑟（Mark Weiser）在施乐公司帕洛阿尔托研究中心（PARC）做的工作。他提出一个假设：当处理器变得足够便宜，以至于可以把它嵌入到各种日常物品中时，会发生什么事情？他的工作围绕这个假设展开，但并没有考虑到网络连接的存在。他创造了“普适计算”这个词来描述这个假设，并且通过他的研究和文章探究普适计算对生活在这样一个环境中的人们意味着什么。

普适计算关注的焦点是嵌入到各种地方的处理器，因此也常被称为**环境计算**（ambient computing）。然而，ambient这个词有“仅存在于背景环境中”的含义，不是指需要我们积极留意的事物，也不是指在某些情况下我们试图移除的事物（例如音频资料中的环境噪声）。

因此，和马克·维瑟一样，我们更喜欢用**宁静技术**（calm technology）这个术语。系统中的物品不会争相吸引你的注意，而是随时准备在你决定关注它们的时候提供便利的功能和有用的信息。

伴随着物理世界中计算设备的激增，出现了各种新的挑战。例如，怎样配置设备，怎样给所有这些设备供电，设备之间怎样互相通信，怎样实现人和设备之间的交流，等等。

功耗和联网方面的挑战是纯粹的技术问题，这些挑战推动了诸如6LoWPAN（www.ietf.org/dyn/wg/charter/6lowpan-charter.html）之类的标准的开发。一个由学者和计算机领域的专业人员等组成的工作组起草了6LoWPAN标准，为的是把下一代因特网协议（IPv6）应用到最简单、最低功耗的有联网能力的传感器上（下一章介绍因特网协议的发展趋势时还会谈到这个话题）。设计该协议的目的是为数量众多的传感器提供足够多的地址并满足其对低功耗的需求。

配置和用户交互方面的问题因为涉及人的参与，显然仅仅想通过技术手段解决是有困难的。好的设计在这些方面却大有用武之地，有助于增加设备的可用性，提高用户的接受度。一个很好的例子就是苹果公司于2001年推出的iPod。iPod不是市场上第一个便携式MP3播放器，

但滚动式转盘用户接口和iTunes软件的配合使用，使得其易用性相当好，成为了一种销量很大的产品。

在孤立的状态下设计一个联网装置，很可能会作出不太理想的设计决策，导致物品或服务在真实的使用环境中表现不佳。借用埃利尔·沙里宁（**Eliel Saarinen**）在设计方面的箴言，我们建议你考虑一下，作为大量的联网装置中的一员，你设计的联网装置是怎么与周边环境互动的。

沙里宁是这样说的：设计一个物品时，总是需要考虑其直接关联的外部环境：椅子放在房间内，房间属于一幢房子，房子存在于一个环境中，环境则在一个城市规划中。除了设计物品，设计服务时也要考虑其所处的物理环境。

对于仅仅是用来感测周边环境或通常用作输入端的联网装置而言，只要它们的活动不需要周边的人参与，应该没什么问题。它们会愉快地收集信息，然后把信息存储到某个在线数据仓库中并进行处理和分析。

一旦装置开始和人发生交互，事情就变得复杂了。我们已经看到，在计算机和手机上，消息通知、弹出窗口和提示音的数量增长迅猛。当我们面对成百上千的新服务和新应用，并且把它们散布在真实世界中的各种物品中时，各种吸引注意力的提示音就是不和谐的杂音了。

针对上述问题，马克·维瑟和约翰·史立·布朗（**John Seely Brown**）提出了一个解决办法，即设计普适计算系统，使物品融入周边环境。这样做的效果是，我们可以一直能感知到它们的存在，直到在需要的时间让它们成为主角。

宁静技术对注意力的中心和外围都有吸引力，并且实际上会在二者之前相互切换。

——“宁静技术设计”，马克·维瑟和约翰·史立·布朗，施乐公司帕洛阿尔托研究中心，1995年12月21日

采用宁静技术的一个很好的例子就是**Live Wire**（有时也被称为**Dangling String**），它是最早出现的物联网装置之一。该装置是艺术家

Natalie Jeremijenko在施乐公司帕洛阿尔托研究中心做客座研究人员时，在马克·维瑟的指导下创建的一个简单装置。一个电机的输出被连到一根8英尺长的塑料绳上。电机的输入被连到以太网上，靠网络数据传输时产生的电信号供电。当网络上有数据包传送时，塑料绳就会被扯动。

在正常的和比较轻的网络负载下，绳子只是偶尔会抽动一下。如果网络过载，伴随着电机运转时发出的独特噪声，绳子会疯狂地抽动。相反的情况，如果没有网络活动发生，绳子就会异乎寻常地平静。两种极端的情况都会引起附近某个人的注意（他已经习惯了绳子的正常行为），他便了解网络出问题了，需要进一步查看一下是怎么回事。

不是所有的技术都需要宁静。一个宁静的视频游戏几乎没什么用，因为视频游戏本来就应该让人兴奋。但还是有太多的设计只聚焦于物品本身及其表面的特征，没有考虑所处的环境。我们必须要学会在设计时考虑周围环境，这样才能充分地驾驭技术，而不是被技术所支配。

——“宁静技术设计”，马克·维瑟和约翰·史立·布朗，施乐公司帕洛阿尔托研究中心，1995年12月21日

Live Wire在高网络负载情况下，电机会发出独特的声响，这是一个有趣的现象。把传送信息的方法扩展到屏幕之外的真实环境中，常常会增加“通知”的维度。在一台计算机上，屏幕更新就是纯粹的基于视觉维度的信息传递。如果要使用其他维度传递信息，也必须要以清楚明白的方式表达。艾德里安的物联网泡泡机Bubblino会对Twitter上的推文进行搜索，当发现与搜索关键词匹配的推文时，就会吹泡泡。和Live Wire类似，Bubblino是一个很好的运用其他维度传递信息的例子，其电机的副效应也是在某事发生时生成听得见的通知。Mint Digital工作室制作的物联网装置Olly（www.ollyfactory.com），把电机和一个有意引入的嗅觉指示器相结合，可以分辨几种不同的社交媒体事件并产生闻得到的通知。

当采用“更好的”技术时，我们需要谨防失去这些类似噪声的“副效应”。若干年前，所有机场和火车站的出发/到达指示牌是用人工翻转的卡片组成的。转轴上有若干卡片，有时会把完整的地名印在卡片

上，有时只是把单个字符印在卡片上，通过翻转卡片显示正确的项目。

在大多数地方，这些卡片式的指示牌已逐步被淘汰，取而代之的是LED点阵显示屏。后者可以很容易地更新为新的目的地，还能实现诸如水平方向滚动显示消息等前者无法实现的功能。但可惜的是，新的显示屏少了一个重要的特性，即老的指示牌更新显示时发出的一连串的噼啪声。结果是，在车站候车的乘客们必须一直盯着显示屏，等待列车到达的消息，而不能像以前那样先忙别的事，仅在指示牌有变化时查看一下。

这不是说显示屏不是正确的选择，仅是手机和平板电脑时代的无意识之选。如果你在开始的设计中选择尽量不使用显示屏，之后你还是会考虑使用它，并认为使用显示屏是最好的解决方案。

有一些有趣的实验，在使用显示屏时，采用了可扫视显示（glanceable displays）的设计。这些屏可被称为次级显示屏，它们不会位于离你很近的位置，而是被放到你可能会放置画框的位置。

显示装置也不都是屏幕。例如，新鲜事物（recently possible）的倡导者拉塞尔·戴维斯（Russell Davies）制作的Bikemap（<http://russelldavies.typepad.com/planning/2011/04/homesense-bikemap.html>），就是把几个LED灯嵌入到一张打印出来的地图中。地图显示的是拉塞尔家附近的区域，每一个LED灯标示了一个伦敦公共自行车租赁系统自行车存放架的位置。如果某一个存放架上有超过5辆的自行车，对应位置的LED灯就会点亮。这个装置被嵌入到一个画框中并且被挂到拉塞尔家的前门附近，他出门的时候只需要扫一眼这个装置，就知道出门往哪边走能找到自行车。

拉塞尔是RIG（Really Interesting Group）工作室的合伙人，这是一个位于伦敦的多学科机构。机构中的其他人，包括他们朋友圈中的一些人，也在宁静技术这一领域不断探索。

他们的工作室里有一套支持AirTunes功能的WiFi扬声器，每个人都可以控制它，用它播放音乐。你在这里工作时，常常想知道正在播放的乐曲的名字，但找不到适当的办法可以在不打扰办公室全体同事的情况下，知道当前的乐曲是谁放的，放的是什么曲子。

为了解决这个问题，他们把一个闲置不用的显示器放置到一个书橱上，不干扰任何人。通过观察网络流量和连接到用来记录播放曲目的last.fm服务，构造了一个显示当前播放曲目及其演奏者的系统。系统的显示屏只在曲目改变的时候更新，其所在位置在所有人的视线之外，因此它不会分散人们对工作的注意力。但如果正在播放的曲目让你分心了，那么这个显示屏正好能满足你的好奇心。

Bikemap也为RIG工作室的克里斯·希思科特（Chris Heathcote）提供了一些灵感。Chris是一名交互设计师，他意识到自己每天早晨都要查看几个手机应用程序，了解天气预报、当天的日程和伦敦地铁的运行情况等信息。他家的门口没有电源插座，因此他把一个信息显示屏放置到了床边。考虑到这个显示屏需要一直开着并且离他睡觉的地方很近，因此持续发光的标准显示屏或其他LCD显示屏都不太合适。然而Kindle阅读器使用的电子墨水显示屏却是一个理想的选择。他利用Kindle的WiFi连接能力和计算能力，把它改造成成了一个自成一体的物联网装置。这个装置只显示一个网页，并且这个网页每隔几分钟就会自动刷新。

这个最终完成的装置被Chris称为Kindleframe（<http://antimega.com/antimega/2013/05/05/kindleframe>），它总是能显示来自于不同网站的最新的聚合内容，这样Chris就能在一天的开始获得他需要的全部信息。

2.2 用魔法作隐喻

在引进不同寻常的新技术或服务时，一个主要的问题是，怎样让人们理解和接受它。对早期的采用者而言，新装置给他们带来各种好处。即便它们的外观看上去有点奇特或用起来有点笨拙，人们也都能欣然接受。然而，为了让这项技术或服务能流行起来，你需要说服大多数人去使用它。

除了自身需要实现特定的行为外，技术常常需要获得**社会**的接纳。很多案例表明，失败的技术和非常成功的技术之间的主要区别是：后者会比前者晚几年出现，此时人们已经变得更乐于接受该技术了。

技术博客的博主Venkatesh Rao想出一个不错的术语，用来解释新技术是怎样被接纳的。他认为人们不会觉得自己所生活的这个世界是持续变化的。如果可以把时间回溯一秒，我们就会知道世界已经改变了。随着时间的流逝，世界已经改变了很多，但这种改变是隐藏在日常生活中，是悄无声息发生的。Rao把这一概念称为**虚构的常态域**（manufactured normalcy field）（www.ribbonfarm.com/2012/05/09/welcome-to-the-future-nauseous/）。

如果一项技术想要被接纳的话，就要设法使其位于虚构的常态域内。因此，成功的用户体验设计师只为用户提供不超出特定的常态域边界太远的体验，即便所采用的底层技术已经远远超出了常态。例如，手机最初是作为不需要用电话线连到特定位置的无绳电话引入市场的，现在，大体同样的技术给用户带来的设备却是便携式因特网终端，可以用来播放影片、存放个人收藏的全部音乐，有时也可以用它打打电话。

便携式因特网终端就是通过用手机作隐喻，设法进入了虚构的常态域。用人们已经理解的事物引入新技术是一个屡试不爽的办法。计算机早期是作为豪华型打字机引入的，而图形用户界面则被比喻为桌面，还有很多这样的例子。

那么，物联网该用什么作隐喻呢？正如我们在上一章所看到的，亚瑟·查理斯·任何足够先进的技术与魔法其实别无二致。鉴于物联网通常会

赐予日常生活中的物品半隐藏的能力，也许魔法和童话故事中有魔力的东西是一个好的隐喻，有助于人们把握各种可能性。

一些物联网项目直接从魔法中获得灵感。例如，John McKerrell的WhereDial就是参照**哈利·波特**中用来追踪韦斯莱家成员所在位置的时钟制作的。韦斯莱家的钟能使用魔力推测每个家庭成员的下落，因而也能了解他们是否遇到危险。相比之下，WhereDial只能依赖技术实现其功能。利用智能手机中的GPS芯片和FourSquare之类的位置签到服务，拥有一个类似WhereDial的装饰品不是什么难事。当你在工作中、旅途中或在就餐场所时，它都可以及时更新位置信息。

案例研究：WhereDial

John McKerrell是一位对地图和地理位置应用充满热情的开发者。他曾用Multimap地图块开发了一个可以在浏览器中随意拖拽的可滑动地图应用（类似于谷歌公司当时刚刚推出的令世人惊叹的地图应用），并把该应用的一个版本展示给Multimap的人看，为此他从这个初创公司得到了一份工作。MapMe.At是他构建的一项网络服务，目的是让人们存储和共享他们的位置。截至2009年，他已经连续好多年经常性地MapMe.At上签到了。

因此，当他参加在自己的家乡利物浦举办的物理计算创意日活动时（碰巧是由本书作者Adrian组织的），提交一个与地理位置相关的项目几乎是必然的了。

在非常流行的哈利·波特系列小说中，有一个韦斯莱家族。这家人有一个可以显示每个家庭成员位置的时钟。John的一位朋友建议他可以用这个钟来形象地显示收集到的位置数据。

John弄到一个传统的老式旅行手提钟，在上面装了一个步进电机，用来驱动钟的机械装置。电机受Arduino电路的控制，而Arduino电路可以和MapMe.At交换信息。钟面上的数字被替换为几个经常出没的地点：家、工作场所、商店、酒馆、餐馆，等等。如果恰巧不在上述地点，则可以显示为“在路上”。把这个钟放到餐具柜上还真不错，不管谁在家里，都可以看到John和他妻子当前的位置（每人对应一个指针）。

之后的几年里，John对钟的设计做了优化，使其成为了一款物联网产品。随着FourSquare等位置服务流行度的增加，钟的软件部分做了升级，可以连接更多的位置服务。钟的结构和外观也有了变化。

用传统的钟做物联网产品有两方面问题：一是找到足够数量的适合改造的钟不容易；二是当某一个指针旋转一圈后，另一个指针才转动到一个新位置，这样就不方便显示两个人的位置。

WhereDial (<http://wheredial.com>) 目前只能显示一个人的位置。另外它也没有采用让指针指向某一位置的方式，而是采用了移动地点标识到特定位置的方式。这个改进后的设计适合实现批量的定制，John可以在他家的工作坊里用激光切割机按需生产足够多的WhereDial。（见图2-1。）



图 2-1 WhereDial

另一些项目也或多或少受到了魔法的影响。

在设计研究领域，魔镜类装置似乎颇受欢迎，尽管其拥有的能力还达不到白雪公主故事中邪恶的皇后用的那面魔镜。人们倾向于在一天开始或结束的时候，在他们预计要使用浴室中的镜子时，用其显示有用的信息。这样，你在早上洗澡的时候，可以查询当前时间、预约活动、交通和天气状况等信息。可以推测，想让魔镜类装置显示你在Facebook上得到的“赞”的数量只是一个时间的问题，这样它就能成为邪恶的皇后那面魔镜（用来查询“谁是世界上最漂亮的人”）的现代等价物。

David Rose曾经管理过一家叫做Ambient Devices的公司，该公司有一些产品可以作为魔镜类装置的范例。根据David Rose关于有魔力的物品的思考，你应该不会对此感到意外。环境感知球（ambient orb）是一个单像素显示装置，用来显示用户选择的度量值（股价、天气预报和花粉计数等）的状态。和外形近似的魔法水晶球一样，环境感知球让你能从远处看到它所传递的信息。

Ambient Devices之后又建造了一把有魔力的雨伞。这把伞可以获取天气预报数据。如果预计要下雨，伞柄就会发光，这样就能在你出门的时候起到提醒你带伞的作用。

这些装置并不像《指环王》中的魔戒一样拥有惊人魔力。它们具备更平凡的魔力，用来更容易地完成任务或让生活变得更有趣。而这正符合我们想阐述的核心观点：利用人们对魔法和童话故事的了解，让这些前所未见的新装置更容易被人们接受。

当然，试图用魔法完成我们能力之外的事情，是有危险的。《魔法师的学徒》（*The Sorcerer's Apprentice*）向我们展示了这一点。当魔法师的学徒试图通过使用他并未完全理解的魔法，使扫帚具备魔力，以此来减轻日常杂务的负担时，扫帚失去了控制。幸好魔法师及时回来，才恢复了秩序。相比之下，到目前为止，我们设计的Roomba自动真空吸尘器看似还是行为正常的。但我们还是要小心，在创建物联网装置时，要避免使其具备用户不易理解的“魔力”和控制界面。

除了要确信物联网装置能按照你的意图行事之外，确保其能为收集到的数据提供防护措施也很重要。

2.3 隐私

谈到信任问题时，让我们感到担心的不仅仅是我们自己拥有的物联网装置。随着有越来越多的感测装置在注视着我们，并且向因特网报告数据，偶然或有意穿越传感器监视区域的第三方带来的隐私问题已经成为一个重要的考虑因素。物联网服务的设计者需要小心权衡这些隐私方面的问题。

2.3.1 保守秘密

对诸如健康护理之类的领域，隐私是一个显然需要关心的问题。我们将在第11章对此做更详细的介绍。然而，看似不涉及隐私的应用也可能泄露个人信息，你应该对此有所警惕并采取必要措施。下面这个例子能很好地说明这一点。

在澳大利亚的一家Westfield购物中心，有一个较早安装了监控设备的停车场。每个停车位用一个Park Assist生产的小型传感器监控，传感器用一个廉价的摄像头判断车位是否已被占用。这些传感器都已连成网络，据推测可以为停车场的管理者提供使用情况的分析。传感器上的灯用来指引司机把车开到空闲的车位。所有这些设施都是有用且无害的。

后来系统引入了一个更高级的特色功能，问题随之而来。购物中心为客人们提供了一款可供下载的智能手机应用程序，方便他们获取购物中心内各种设施的更多信息。这个应用程序提供了“我的车在哪里”（Find My Car）这一功能选项。选择此功能后，程序提示你输入车牌号的前几个字符，然后会返回给你四幅可能匹配的小图片。这些小图片是通过使用OCR（光学字符识别）软件，对购物中心服务器上存储的传感器数据进行处理后得到的。

这些小图片只是些缩略图，其清晰度只够识别出哪辆是你的车，车牌看上去模糊难辨，不能用来做其他事情。然而，安全专家Troy Hunt却发现，这个程序的实现方法有问题，有可能泄露个人信息

（<http://www.troyhunt.com/2011/09/find-my-car-find-your-car-find.html>）。

使用现有的一个非常简单的软件，就可以对这个手机应用程序发送给服务器程序的请求消息进行观察。Troy发现，该消息是一个非常简单的未经加密的Web请求消息。原始的请求URL包含若干参数，既包括要搜索的字符串，也包括诸如要求返回的结果数之类的信息。

这个请求消息返回的是一个数据块（采用的是容易解析的标准的JSON格式），其中不仅包含四幅图片的URL，也包含许多额外的信息。可以据此推测，在本例中，对于Web服务的开发者来说，返回全部相关的数据比只返回所需的数据要容易。这些额外的数据中，不仅包括每个传感器单元的IP地址等一般信息，而且也包括每一辆车完整的车牌号和泊车时间等重要信息。

Troy发现，通过改变查询参数，可以获得远多于4个的匹配结果。而省略掉要查询的车牌号字符串也是可能的，这意味着只要他乐意，随时可以通过一次Web请求，下载到全部2550个停车位上所泊车辆的车牌号列表。

虽然上述所有的数据都可以通过公开的方式获得，比如在停车场的入口处持续监视车辆的进出，但是在计算机上创建一个脚本并以固定的时间间隔进行查询，就能轻易获取所有的数据。

收到针对该问题的警告后，Westfield和Park Assist马上就禁用了这一特色功能，然后通过Troy协作，构建了一个更好的解决方案。然而，要不是Troy慷慨大度地让他们及时注意到问题所在，最终结果就不好说了。

提供服务时，不要分享超出所需的信息。

维基解密的创始人朱利安·阿桑奇（Julian Assange）说过：“保守秘密的最好方式是从来不知道这个秘密。”（<http://www.pbs.org/wgbh/pages/frontline/wikileaks/interviews/julian-assange.html>）如果你能一开始就避免收集和存储这些数据，就没必要担心意料之外的泄密。

在当今时代，决不以明文的形式存储密码是一个业内常规。你可以考虑应用标准的密码加密机制，例如单向散列，把密码转变为另外一组数据。Peter Wayner在《透明的数据库》（*Translucent Databases*）一

书中就推荐采用单向散列技术。如果你不需要把数据恢复到其最初的形式（即只需要其独一无二并且与同一组数据关联），那么就不要在数据库中存储可恢复为初始形式的身份识别数据，保存一个用单向散列处理后的版本就可以了。这样做了以后，数据的拥有者仍然能找到他们的数据（因为他们能再次提供原始的身份识别数据并对其做散列运算），对数据做统计分析和报表之类的工作也不受影响。

散列 (Hash)

单向散列是一种加密技术，用来把任意大小的数据块浓缩为一段称为“散列值”的固定长度的信息。之所以被称为单向散列，是因为在给定散列值的情况下，想反方向计算出原始形态的数据是很困难的。散列算法都是这样设计的，即输入数据只要稍有不同，就会导致其输出的散列值有明显的差别。

当你想验证两个数据块是否完全相同，又不想为了做比较保存这些数据时，散列算法的上述特性就非常有用。如果你想做比较的数据块非常大，或者你不想以原始形态保存数据，那么使用散列算法正合适。

密码散列的最常见用途是密码验证。服务提供商不储存用户的原始密码，而只存储密码的散列值。当需要鉴别用户身份时，可以根据用户的输入重新计算散列值。如果计算结果和存储的散列值匹配，则服务器端有理由相信该用户提供了正确的密码。

应用散列算法之前，先对密码做加盐处理是一个好习惯。所谓加盐，就是在计算散列值之前，给密码添加一些随机的、无需保密的被称为盐（salt）的额外的文本。盐和散列值存储在一起，当需要验证新提交的密码时，服务端可以把该密码和盐重新连接起来。攻击者可能会设法获取散列值的一个副本，再通过把它与一个预编译的散列值字典做比较，以此来获取密码。对密码做加盐处理就能防止这种攻击。

2.3.2 谁的数据

虽然被部署的传感器的数量在不断增加，但被收集的数据应该归谁所有一直不太明确。例如，部署在广告牌上面的摄像头能够查看人们是否在看各种不同的广告。这些数据是属于安装摄像头的公司，还是属于正在看广告和普通民众？城市计算（urban computing）的主要实践者Adam Greenfield对此有一个很有说服力的论点。他认为，在公共场所中，这些数据是由公众生成的，因此公众至少应该拥有平等的知情权，也有权使用这些数据。

（<https://speedbird.wordpress.com/2012/12/03/the-city-is-here-for-you-to-use-100-easy-pieces/>，参见第67条。）

在私有物业里，你可以更容易地宣称公众无权拥有相关数据。主张拥有这些数据的人可能是该物业的业主，而不是安装摄像头的人。诸如购物中心之类的很多地方，尽管被私人拥有，但看上去感觉像是公共场所。在这些地方又怎样界定数据的归属权？

2012年夏天，开放物联网大会（<http://openiotassembly.com/>）的参会者们在开会讨论此类问题时，创造了**数据主体**（data subjects）这一术语。对于数据所涉及的当事人来说，不管那些用来收集数据的传感器和用来安装传感器的物业是否为他们所拥有，他们都是数据主体。数据主体应该享有什么样的权利目前尚不明确，但这是一个值得进行更多探讨和关注的领域。

2.4 联网装置的Web思维

当你考虑物联网装置网络方面的问题时，从现有网络部署中获取经验教训和设计准则可能是有帮助的。我们明显可以借鉴一下万维网及因特网本身的设计准则。毕竟，**物联网**这一术语将来会显得有点古怪，因为人们将来肯定会认为，因特网上不仅连接着计算机和电话，而且还有大量的能够上网的物品，他们丝毫不会认为这种情况有多么奇怪。你应该致力于养成Web思维方式，创建**属于** Web而不仅仅是**附着**在Web上的装置。

在TCP协议规范的一个早期的版本中（RFC761，<http://tools.ietf.org/html/rfc761#section-2.10>），乔恩·波斯特尔（Jon Postel）就提出，“发送时要保守，接收时要开放”（Be conservative in what you do, be liberal in what you accept from others）。从此之后，这个健壮性准则就广为人知，并通常被称为**Postel法则**。在设计或构建必须和其他服务交互的任何系统时，牢记Postel法则是有益处的，特别是当与其交互的其他组件并不是由你构建时。

2.4.1 小块松散组合

即便一个服务的所有组件都是你构建的，让这些组件之间的耦合度不要太高也是有意义的。因特网之所以能繁荣发展，正是因为不存在可以完全控制全网的中心位置。因特网正是一种遵循了“小块，松散组合”（small pieces, loosely joined）准则的包含了各种服务与设备的集合。

对服务的设计者来说，这意味着服务的每个组件都应该能做好单一的一件事情，并且不能太依赖它所使用的各个独立组件。要尽可能地让组件更为通用，使其能够为那些也需要相似功能的其他系统服务。这样一种设计，能让我们通过对现有组件进行重用和稍作改动，更方便地构建出在最初设计的系统投入使用时未曾想到的新功能。

尽量使用已有的标准和协议，而不是自己另搞一套。如果人们可以利用标准库和相关技术与你的系统交互，或者在你的系统之上构建应用，那么即便在代码或硬件的优雅程度或效率方面有一些损失，也是

可以接受的。例如，Twitter的设计师们实现搜索功能时，在查询结果的表示方式上，选择包含了一种更适合机器识读的方式，即标准的Atom信息聚合格式（<http://tools.ietf.org/html/rfc4287>）。我非常肯定，他们没有料到这些查询结果会被Arduino使用。Arduino能利用这些数据触发Bubblino泡泡机，每当收到一条新的推文时，泡泡机就会吹泡泡。

Bubblino可以通过理解和使用Atom提要（feed）来寻找新推文，因此只需对它略作修改，就可以用它来监控任何其他支持Atom提要的被监控对象。如此看来，用Bubblino监控某个博客是否有新博文发表是相当容易的。对任何其他你想用吹泡泡的方式表达的通知消息，都可以表示为Atom提要。大多数Web编程语言有现成的函数库，用来生成适当的Atom提要。

这意味着，如果我们的客户想用Bubblino监控开发者的行为，如他们的某位开发者提交了代码并且这些代码没有通过自动化测试，是相当容易的。他们只需要编写一个小型的Web服务，用来获取测试结果并生成Atom提要。一旦他们的Bubblino被配置为使用这个新的Web服务，它就能在测试未通过时吹泡泡。

2.4.2 因特网上的一等公民

松散耦合还有一个深层含义，那就是让装置尽量成为因特网上的一等公民。这里的“一等公民”指的是：你要和因特网上的其他成员保持一致，尽可能使用相同的协议和规范。

在项目初期，人们倾向于选择折衷方案和容易实现的协议。的确，很多中间件供应商鼓励这种做法，并且声称此类低功耗终端的能力是有限的（或者因为没有足够的处理能力或内存，或者受限于所用的网络）。这种说法有一定道理，但也不可尽信。过去20多年形成的一条有价值的经验规则是：IP协议会渗透到任何地方。所以，我们没有理由认为，该规则不再适用于物联网。

少数情况下，例如对于超低功耗传感器，现有的协议不能直接使用。有一个更好的解决方案：和你的合作伙伴一起去修改现有标准，或者在传统的标准框架内，创建能解决问题的新开放标准。

移动Web的演进是一个让人引以为戒的好例子。手机刚可以连接因特网时，人们认为用手机直接访问Web服务器实在太困难了，因此他们开发了一套新协议，即无线应用协议（WAP）。手机可以访问专门为其定制的WAP网站，或者通过WAP/Web网关服务器来访问标准的Web网站。但这要求WAP网站开发者们先学习一套新技术，因此新协议推广得并不顺利。而没有可以访问的WAP网站，用户接纳移动Web的步伐就更缓慢了。

随着技术的逐步演进，在手机上已经可以使用标准的Web协议。尽管网页的显示效果并不完美，但已经好到用户愿意开始使用的程度了。因为用手机直接访问Web站点的人不断增多，开发者们看到“便于移动用户使用”（mobile-friendly）特性的需求。鉴于这些特性可以让开发者们利用他们已经熟悉的工具进行开发，所以久而久之，移动Web就演变为广义的Web技术的一个方面了。

2.4.3 优雅降级

因为因特网广受欢迎，并且允许各种设备和服务在其上运行，所以不同种类的因特网终端拥有的能力差异很大。也正是因为这个原因，几乎不可能构建出能够适用于所有这些终端的服务。然而，若干设计模式的演进减轻了这一问题的影响。

第一种技术是承认大量不同类型的装置很可能带来问题，并且在设计系统的时候就考虑到这一点。如果你需要提供一个在装置间传递数据的格式，别忘了在其中添加一个用来在不同版本的格式之间做区分的机制，最好做到能让旧装置仍然能读取新版本数据格式中的主要内容。这一般被称为**后向兼容**。虽然随着时间的推移，数据格式中会有一些没用的内容（因为一些特性只适用于过时的装置），但这项技术能极大地延长用户装置的生命期，提高装置的利用率。HTML的格式允许任何客户端忽略任何不能识别的标签（<>中的文本），以此来实现后向兼容。因此，在不影响旧版本解析器使用的情况下，新版本的HTML格式能够添加新的标签。HTTP协议实现后向兼容的技术则略有不同：通信双方分别指明各自支持的协议版本号，支持高版本协议的一方会避免在会话过程中使用任何新特性。

另一种常用技术被称为**优雅降级**。该技术旨在为能力足够强的客户端提供全部特性的用户体验，而对于能力不足的客户端，则可以自动降级——可能有若干个级别，提供部分特性的用户体验。优雅降级可以跨越不同的实现技术，并且这种情况还十分常见。

开发人员在实现诸如Twitter和Gmail之类的富Web应用时，会想使用现代浏览器支持的各种高级的JavaScript特性。优秀应用会在使用某些特性之前检查其是否可用。如果这些特性不可用，该应用会限制自己仅使用更简单、更常用的JavaScript代码。例如，提交表单前仍然会验证表单内容，但不再调用服务器端的方法以实现自动完成功能。如果JavaScript完全不被支持，则还能回退到基本的HTML表单。虽然此时的用户体验不能和支持全部JavaScript特性时相比，但总好过完全不能用。

在设计联网装置时，除了可以通过使用几种功能相同的技术实现优雅降级外，也能够把优雅降级应用到装置本身，实现一定程度的故障容错能力。随着装置数量的激增，装置出现某种故障的可能性也在增加，这意味着：当有些部件不能正常工作时，让装置继续具备使用价值是很重要的。假设一个早期的有因特网接入能力的冰箱不能再连接到你的WiFi基站，因为它只支持IPv4，而因特网世界已经迁移到IPv6，你仍然能够用它的触摸屏写消息，能够把U盘插到冰箱上并用触摸屏查看存储在其中的照片。如果连触摸屏也坏了，你还是能够继续使用冰箱的冷藏冷冻功能。

2.5 功能可供性

在《设计心理学》（*The Design of Everyday Things*）一书中，唐纳德·诺曼（Donald Norman）就**功能可供性**（affordances）给出了如下的定义：

功能可供性为物品的操作提供了明显的线索。平板是用来推的，旋钮是用来转的，狭长的方孔是用来插东西的，球是用来抛掷或拍的。如果物品的功能可供性被合理利用，用户一看便知如何操作，无须借助任何的图片、标签和说明。复杂的物品也许需要操作说明，简单的物品应该不需要。如果简单的物品也需要用图片、标签和说明来解释其操作方法，这个设计就是失败的。

我们向所有对设计感兴趣的人推荐这本优秀的书籍。不过，该书关于门的设计的可供性章节可能会破坏你对建筑和人互动方式的认知，因为在现实生活中你几乎每天都会碰到考虑不周的设计，看完书你会郁闷的。

物联网的应用步伐在加速，越来越多的城市、家庭和环境将充斥各种技术。这些新增的行为和能力将带来额外的复杂性，成功的物联网装置和服务设计师需要设法抵消其影响。

被赋予到物品上的很多新能力本质上不可见，或者看上去并不明显，这给直觉设计带来困难。对于数字化的有魔力的物品，功能可供性是指什么呢？

我们怎样向用户传递信息，让他知道某个物品可以和云端通信？或者让他知道这个装置支持诸如RFID之类的短距离通信方式？怎么看得出一个玩具可以测量温度？玩具震动的时候又表示什么含义？你怎样能知道本地的公交车候车厅是否在监视你？或许更重要的是知道，它为什么要监视你？

解决这些问题的一个重要的开端，就是让物件现有的功能可供性得到加强。那些没有察觉装置有任何附加功能的人，仍然能继续使用这个

装置，只当附加的功能并不存在。虽然这个原则听上去是常识，但它经常因成本或设计难度等原因被人们放弃。

例如，一个“非智能的”（**dumb**）的灯光调节开关通常设计为旋钮样式，使用户能够精确地控制亮度。一旦把它接入家庭自动化系统，灯的亮度就可以被远程控制或自动控制。这样的话，同步旋钮的位置和灯光亮度级别就有一定难度，因此经常会导致旋钮被替换为若干按钮。这样做的结果是，用户同时失去了快速大范围改变亮度和小范围精确调整亮度的能力。解决此问题的一个好办法是使用电动电位器，其实就是很多音响设备中使用的音量旋钮。用户仍然可以用传统的方式调整这个旋钮，而任何远程实现的亮度改变都会立刻反映为旋钮位置的变化。

对于不可见的交互，或者因为使用了无线通信，或者因为是通过学习过的姿势触发，事情要更棘手一些。不过，我们仍能通过对物品的外形进行适当设计，鼓励正确的行为。**RFID**技术本身并不要求物品具备卡片形式的外观，但卡片形式的设计可以引导用户采用正确的交互方式，例如在乘坐伦敦地铁时，把他们的**Oyster**交通付费卡与形状相似的读卡器表面轻轻接触。

设计物理接口时，类似的规则也适用。不要给人们熟知的连接器添加不熟悉的行为。例如，不应该使用3.5毫米的音频插口给系统供电，尽管有点另类的用它传输数据可能还行。当你设计一个全新的连接器时，要考虑一下怎样防止用户连接时搞错方向。**littleBits**

（<http://littlebits.cc/about>）在设计模块化的可以接插在一起的电子线路构件时，就碰到了这个问题。因为产品的目标用户是初学者，所以他们想寻找一种方法，可以毫不费力、简单明了地把构件连到一起。他们的解决方案不错，做到了这一点。他们使用磁铁鼓励正确的连接，同时还阻止了不正确的连接。

2.6 小结

本章内容将会让你对新兴的物联网领域有更深入的理解。本章介绍的若干方法，在你设计物联网装置时，也将在思维方式层面为你提供一些指导。

你不仅要考虑装置“应该怎样工作”这样的技术细节，也要考虑怎样把它融入用户生活的大环境中。本章通过一些实例说明了这一点。对于手机应用程序，一旦切换到另一个程序，先前的那个程序就看不见了。与此不同的是，物联网产品会在真实世界中占据物理空间，不会因为用户关注焦点的转移而消失，你要考虑到这一点。

你也要注意不要泄露用户不希望泄露的任何信息。物联网是一个充满各种可能性的新领域，给我们提供了让生活更快乐和更丰富的机会。但我们在推进物联网发展的同时，要注意不要疏远不太懂技术的人，别让他们感到恐慌。

适当使用诸如魔法和童话故事之类的切入点，有助于让不太懂技术的人接纳物联网。在计算机相关的增强功能失效后，可以优雅降级，仍然可以用人们习惯的方式使用的系统，也有助于做到这一点。

虽然本书的内容才刚刚展开，你已经通过一些介绍，例如，对Natalie Jeremijenko的Live Wire的描述，或在优雅降级一节提及的HTTP协议的版本，接触到了网络的不同方面。

对于一本物联网相关的书籍，这没什么可惊讶的。但我们并不假设你能完全理解网络是怎样工作的，或它到底能做些什么。下一章我们将介绍因特网的常用协议（包括协议的概念），以及协议之间如何互相关联，来让你更好地理解因特网的工作方式。

第3章 因特网原理

如果你正在阅读本书，说明你很可能经常使用因特网，用它来浏览网页，阅读和发送电子邮件，听音乐。但对因特网的熟悉程度是分层次的。每天使用因特网只是理解因特网的第一步，开发能连接因特网的软件或物品是一个层次，开发和调试支持因特网本身运作的软件则又是另一个层次。

本书的作者们在对因特网的熟悉程度上也是有差异的。也许你和艾德里安一样，已经把完整的TCP/IP协议栈实现了若干次；也许你和哈基姆一样，并不完全明白TCP/IP协议栈是什么含义。

但不管你是开发者、工程师、艺术家，还是企业家，对因特网的不同组成部分和技术有一个高层次的了解，将有助于你理解物联网应用的前景和当前的局限。

本章是一个不太深入的简短综述，旨在提供良好的可读性，而不是追求内容的全面、完整。让我们从一个例子开始，看一下现实世界中的远距离沟通是怎样实现的，之后再将其与信息在因特网这个虚拟世界中的传输方式做个比较。

3.1 因特网通信概览

假设你想给本书的作者们发送一个消息，但你并不知道我们的邮政地址，也没有办法查询我们的电话号码（本例假设没有因特网可用）。

你记得我们来自英国，而伦敦是英国最大的城市，因此你给住在伦敦的表弟**Bob**寄送了一张明信片。

你表弟看到这张明信片是寄给两个疯狂的硬件与技术爱好者的，因此他把明信片放入信封，然后把这封信放到了伦敦黑客空间（**London Hackspace**）。那里的人很可能知道如何处置它。

London Hackspace的**Jonty**拿起信，看到它是给利物浦的某些人的。就像所有的伦敦好市民一样，**Jonty**从来不会去沃特福德以北的任何地方，但他记得曼彻斯特也在北边。因此，他打电话到曼彻斯特数字图书馆（**MadLab**），把信的内容读给对方，并且说道：“这个消息是给在利物浦的艾德里安和哈基姆的，能麻烦你把消息传过去吗？”

MadLab的人会继续打听是否有人认识我们两个，结果发现**Hwa Young**认识我们，就把消息传给了她。于是，在**Hwa Young**下一次来利物浦的时候，她顺便就把消息告诉给我们了。

3.1.1 IP

上面这个故事恰好描述了**IP**协议（**Internet Protocol**）的工作原理。数据以分组（**packet**）的形式从一个装置发送到另一个装置。数据分组包含一个目的地址和一个源地址，两者都符合一种标准化的格式（“协议”）。正如上例中消息最初的发送者一样，发送装置并不一定能提前知道到达接收方的最佳路径。大多数情况下，数据分组不得不通过若干被称为**路由器**的中间节点，才能到达目的地址。正如上例中用到了电话、邮政服务和人工投递一样，底层网络也并不一定是固定不变的，数据分组可能会通过各种途径传送：有线或无线网络、电话系统或卫星链路。

在上例中，明信片被放到了一个信封中传送。对于因特网数据分组，也是类似的情况。**IP**数据分组由实际的数据块与主机名及地址等附加信息组成，这些附加信息的作用就相当于你写到信封封皮上的内容。如果一个**IP**数据分组在你本地的有线网络上通过以太网网线（把家用宽带路由器或办公室的局域网连到一台桌面**PC**的线缆）传输，则需要对整个数据分组进行封装（相当于放入信封），形成**以太网帧**。以太网帧增加了额外的信息，用来实现把数据分组送达你的计算机所需的最后几个步骤。

当然，你的表弟**Bob**有可能并不知道伦敦黑客空间，消息也许会滞留在他那里。你没办法知道消息是否送达到接收方。**IP**协议正是这样工作的：它并不能确保信息不丢失，并且只能发送单个数据分组可以容纳的信息。

3.1.2 TCP

如果你要传递的信息较长，一张明信片写不下，或者想确保消息到达目的地，又该如何处理呢？

假如有这样一种约定：只有对于用绿色墨水写就的明信片，我们才十分关心它的送达正确性，那一切就简单多了。另外，我们可以给这种明信片编号，按编号次序将信息依次发送出去，从而实现较长信息的传送。收信人收到一封封明信片后，再依照编号把这些明信片上的消息片段按编号顺序拼在一起，即便有些明信片没有按顺序送达（也许你好几天时间都在按次序写明信片，而**Bob**在收到第五张明信片那天，正好要去利物浦，因此，这张明信片就不需经过伦敦黑客空间或**MadLab**的中转，直接由**Bob**捎给我们），收信人也能够向发信人发送通知，告诉发信人哪些明信片已经收到，以便发信人能够重新发送丢失的明信片。

传输控制协议（**TCP**）的工作机制与上面的描述大体相同。**TCP**是因特网上的传输协议，建立在基础的**IP**协议之上，增加了序号、确认和重传机制。这意味着用**TCP**可以发送任意长度的消息，并且能在一定程度上保证将信息完整地送达到目的地。

因为TCP和IP的组合非常有用，很多服务，如电子邮件，以及万维网（WWW）用于传输信息的HTTP协议，都是依次建立在这个组合之上的。

3.1.3 IP协议栈

由于TCP和IP的组合无处不在，所以我们经常把以TCP和IP为基础、分层堆叠在一起的一整套协议（即“协议栈”）简称为“TCP/IP”。在这个协议栈中，每一层都是建立在它下面一层的功能之上（如图3-1所示）。下面就来介绍TCP/IP的各种层级。

- 位于**链路层**的底层协议，负责管理一条网络链路上的信息传送过程。链路层协议的管理对象包括以太网网线、WiFi、电话网络，甚至还包括诸如IEEE 802.15.4之类的短距离无线通信标准。此类短距离无线通信标准可以实现数据在个人区域网络（PAN, Personal Area Network）上的传送，即在个人随身携带的装置之间进行数据传送。
- **网际层**位于各种链路之上，屏蔽了链路层的各种细节，实现了对一个简单的目的地址的支持。
- 位于**传输层**的TCP建立在IP之上。通过对IP协议的扩展，TCP能够对消息传递实施更为复杂的控制。
- **应用层**协议包括了与获取网页、发送电子邮件和网络电话有关的协议。其中，HTTP是最常用的Web协议，它实际上也适用于物联网装置之间的通信。我们将在第7章对MQTT之类的标准进行简要介绍。

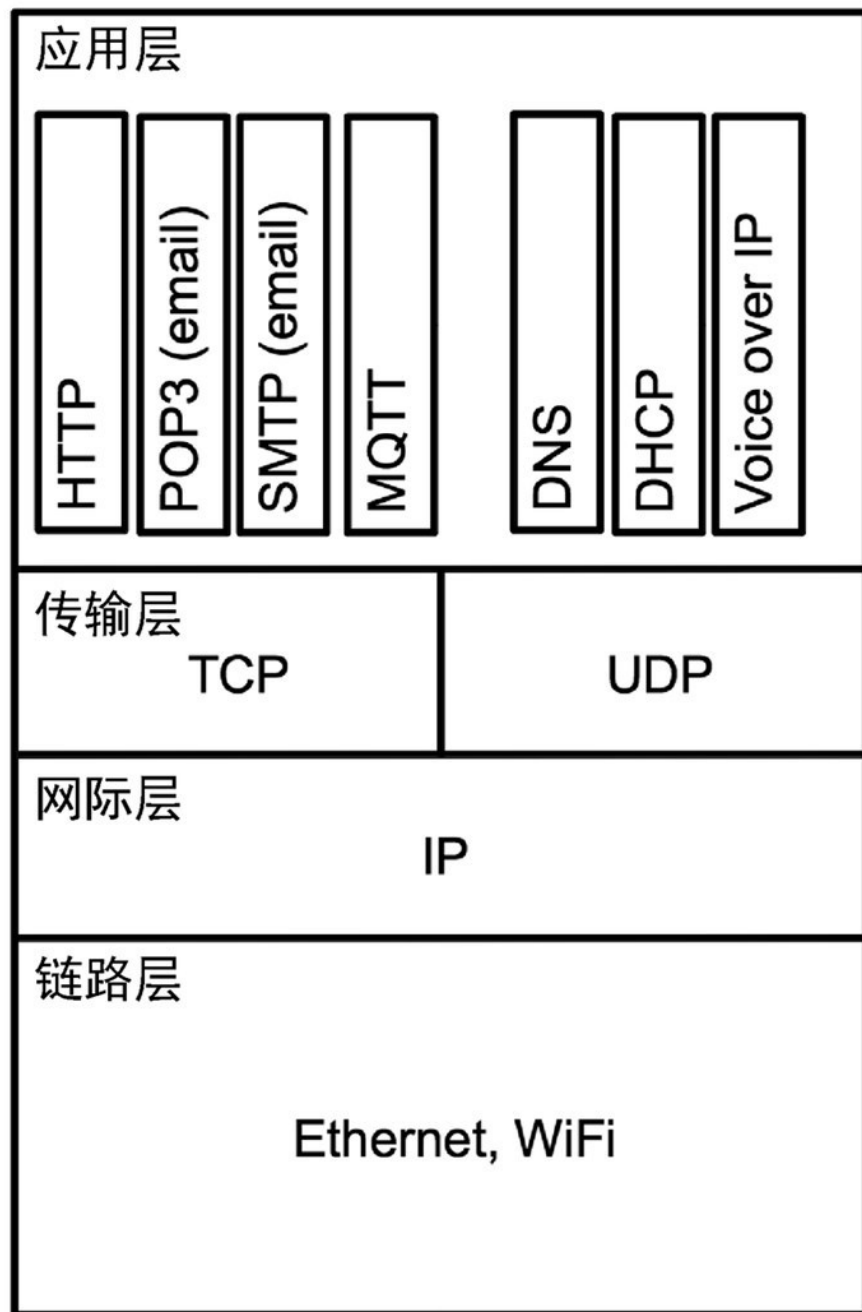


图 3-1 IP协议栈

3.1.4 UDP

如你所见，TCP协议不是传输层唯一的协议。不同于TCP协议，但与IP协议一样，在使用UDP协议时，每个消息不一定能到达目的地，没

有握手或重传发生，也没有等待下一编号消息的延迟时间。这些限制使得物联网装置在实现很多任务时，更愿意使用**TCP**协议。

然而，这些限制和功能的缺失也使得**UDP**更适用于流数据类的应用。此类应用能妥善处理小错误，但不希望有长的延时。**VoIP**（**Voice over IP**），即基于计算机的电话通信，诸如**Skype**之类，就是此类的一个例子。丢失一个数据分组可能会导致声音质量受到轻微影响，但如若等待若干数据分组按顺序到达，则可能会导致语音出现较大抖动而不容易让人听懂。一些非常重要的协议，例如**DNS**和**DHCP**等，也会选用**UDP**作为传输层协议。这些协议能提供和网络中设备的发现和解析有关的公用基础服务。我们将在下一节中详细介绍。

3.2 IP地址

我们之前提到，IP协议知道源装置和目的装置的地址。但这个地址是由什么构成的？请看一个典型的人类（以霍比特人为例）地址：

比尔博·巴金斯
袋底洞，袋边街
霍比屯
夏尔郡
中土世界

然而，在底层的计算机网络世界里，处理数字要比处理文字容易得多。因此，IP地址是用数字表示的。对于第四版的IP协议（IPv4），可获得的IP地址差不多有43亿——准确地说说是4 294 967 296，即 2^{32} 。虽然数字形式的IP地址便于计算机使用，但不适合人类识读。因此，IP地址通常被表示为由点号分割的4个8位数字的组合（从0.0.0.0到255.255.255.255）——例如，192.168.0.1（通常用作家用路由器的地址），8.8.8.8（谷歌公司的一台DNS服务器的地址）。

这种“点分四组”形式（dotted quad）的数字仍然完全等价于一个32位的数字。除了便于人们记忆，通过把某些地址块聚合起来，这种表示法也便于人们推测与某一地址相关的信息。例如：

8.8.8.x——分配给谷歌公司的若干IP地址区间之一。
192.168.x.x——指派给私有网络的一个地址区间。家庭或办公室的网络路由器可能会在这一范围内分配IP地址。
10.x.x.x——另一个私有地址区间。

因特网上的每一个装置至少需要一个IP地址。也就是说，每一台计算机，每一台联网的打印机，每一部智能手机，每一个物联网装置，都需要一个IP地址。如果你已经有了树莓派、Arduino板，或其他任何将要在第4章和第5章中介绍的微控制器，它们也都应该有自己的IP地址。考虑上述情况，4亿IP地址突然间就好像不够分配了。

诸如192.168.x.x这样的私有地址区间缓解了IP地址不够用的问题。家庭或办公室的网络可以只有一个公有的IP地址，但却可以把从192.168.0.0到192.168.255.255这个范围内的全部IP地址（ $2^{16} = 65\,536$ 个地址）分配给不同的装置使用。

针对这一问题的更好的解决方案是使用下一代IP协议——IPv6，对此我们将在本节后续部分介绍。

3.2.1 DNS

虽然计算机能够轻松地处理32位的数字，但即便采用点分四组的表示法，对大多数人来说，也很难记住这些数字。域名系统（DNS）能帮助我们记忆力欠佳的大脑，让我们能畅游因特网。通过使用万维网、电子邮件或其他网络服务，我们对下面这样的域名应该并不陌生：

```
google.com  
bbc.co.uk  
wiley.com  
arduino.cc
```

每个域名都包含一个像.com 或.uk 这样的顶级域名（TLD）。顶级域名又可以进一步划分为.co.uk 、.gov.uk 等子域名。顶级域名知道去哪里可以找到它所包含的子域名的更多信息，例如，.com 知道要去哪里寻找google.com 和wiley.com 。

这些子域名进而知道去哪里找单独的装置或服务。例如，子域名.google.com 的资源记录知道要把对下列域名的访问指向哪里：

```
www.google.com  
mail.google.com  
calendar.google.com
```

这三个域名都是可以直接识别的网站名，也就是说，你可以在浏览器中输入这些域名，例如，<http://www.google.com/>。

DNS也能够指向因特网上的其他服务，例如：

```
pop3.google.com—用于接收Gmail邮件  
smtp.google.com—用于发送Gmail邮件  
ns1.google.com—谷歌公司诸多域名服务器中的一个
```

配置DNS就是改变几个设置项的问题。你的域名提供商（卖给你域名的那个公司）经常会提供一个控制面板界面，用来改变这些设置项。你也可以使用自己的权限域名服务器。对于子域名roomofthings.com，其设置项中可能包含如下的资源记录：

```
book A 80.68.93.60 3h
```

该资源记录的含义是：**book.roomofthings.com**（本书博客所在主机的域名）这个域名地址对应的IP地址是**80.68.93.60**，并且这一对应关系在接下来的3个小时内不会改变。

3.2.2 静态IP地址分配

怎样能获得一个IP地址？如果你从因特网服务提供商（ISP）那里购买了服务器托管业务包，你通常会获得一个单独的IP地址。而ISP本身拥有的是一个可供分配的地址块。地址块代表不同大小的地址区间，通常被划分为8位、16位和24位三类：

A类——从0.x.x.x开始 B类——从128.0.x.x开始 C类——从192.0.0.x开始
--

C类地址区间只有8位（256个地址）可供分配，而A类地址区间可供分配的地址要多得多，因此只能把A类地址块分配给非常大的因特网机构。把地址空间严格分割为三类的做法不是非常有效率。每个机构都想保留足够多的备用地址，以备将来的扩展，但这也意味着很多地址会被闲置不用。随着接入因特网的装置数量激增（本章一直在强调这一点），从1993年开始，分类方式的地址划分方案被**无分类域间路由（CIDR）**代替，后者允许你精确地指定IP地址中固定不变的位数（参见RFC1518和RFC1519，<http://tools.ietf.org/rfc/>）。因此，前面提到的第一个A类地址块等价于**0.0.0.0/8**，而**208.215.179.0/24**则是一个C类地址块。

例如，前面已经提到，谷歌公司拥有**8.8.8.x**这个地址区间（用CIDR表示法表示就是**8.8.8.0/24**）。谷歌公司从这个地址区间中，选择**8.8.8.8**作为其一台公共域名服务器的地址。之所以选择**8.8.8.8**，就是因为它好记。

不过，在很多情况下，系统管理员只是简单地按顺序分配主机编号。管理员会对分配出去的地址做记录，并且会更新DNS中的资源记录，使得对域名的解析能指向这些地址。我们把此类地址称为是**静态**的，因为一旦分配到了地址，在没有人为干预的情况下，主机的地址将一直保持不变。

再看一下你家里面的网络。每一次把PC机的网线接头插到路由器上，把笔记本电脑或手机接入无线网络，或者打开有联网能力的打印机时，这些装置都需要获得一个IP地址（通常在**192.168.0.0/16**这个区间内）。你可以自己按顺序分配地址，但使用家庭网络的人通常不是专业的系统管理员，不会对地址的分配情况做详细的记录。假设你弟弟以前一直在用**192.168.0.5**这个地址，但他好久没回家了。现在你弟弟回来了，却发现你的新激光打印机占用了这个地址，结果他就连不上网了。

3.2.3 动态IP地址分配

幸运的是，我们通常并不需要为每一个要联网的装置选择IP地址。当把笔记本电脑、打印机，甚至一台能关注推文更新情况的泡泡机连入网络时，这些装置可以使用动态主机配置协议（DHCP）向网络申请一个IP地址。当这个装置试图接入网络时，它不再检查自己内部的地址配置，而是会给路由器发送一个消息，申请获得一个地址。路由器随即给它分配一个地址。这个地址不是一个可以无限期使用的静态IP地址，而是根据当前可用的地址资源，从中**动态**选取的一个临时的、有一定租期的地址。如果路由器重新启动，租约到期，或装置关机，某个其他的装置可能会获得这个IP地址。

这意味着你不能简单地把一个DNS记录指向一个使用DHCP的装置。一般情况下，你可以认为在一次特定的工作会话过程中，IP地址很可能不会改变。但是，你不应该硬编码（hard-code）IP地址。因为，当你下一次使用它时，它可能已经发生变化了。

即便是诸如Arduino板之类的最简单的计算装置也可以使用DHCP，我们将在第5章中介绍。虽然Arduino的以太网库允许你配置一个静态的IP地址，但你还是可以使用DHCP动态获取IP地址。在开发阶段使用静态地址也许是个不错的选择（如果你是唯一使用这个静态地址的人），但对于团队开发的情况，或者这个装置是准备分发给在任意一个网络中的其他人的，你几乎肯定需要使用动态IP地址。

3.2.4 IPv6

当初实现IP协议标准化的时候，几乎没有人能预见到IPv4提供的43亿地址会被分配得这么快。这样看来，物联网的增长只会加速这一趋势。如果你把手机、手表、MP3播放器、增强现实太阳镜、远程医疗和运动监测装置都接入因特网，你个人就已经占据6个IP地址了。也许你还有一个专用的用于微支付的电子钱包服务器，一个包括了你的联系方式和博客的个人Web服务器，一个或多个记录你日常行为的网络摄像头。你也许使用的不是单一的健康监测装置，而是在身体各处的若干装置，包括测量体温、心率、胰岛素水平等刺激源的传感器。

在家中，你首先会把所有的电子设备接入网络。出于安全考虑，还会在每扇门和窗户上都装上传感器。另外，为了检测老鼠和甲虫，可能还要装上许多灵敏的声音传感器。为了提高效率，可能还有其他用于检测温度、湿度和气流级别的传感器。很难预测一个家庭在不久的将来拥有的物联网装置的数量，到底是几十个、几百个，还是几千个？

IPv6的地址长度是128位，通常把一个IPv6地址表示为8组数，每组包含4个十六进制数。例如，**2001:0db8:85a3:0042:0000:8a2e:0370:7334**。IPv6的地址空间特别巨大（ 2^{128} ），你甚至可以给地球上的每一个人都分配海量的、和IPv4地址空间中的全部地址一样多的地址。即便真这样做了，相对于IPv6巨大的地址空间，所分配的地址数量也是很少的。

IPv6这个新标准在上世纪80年代就在讨论酝酿了，最终于1996年发布。但即便是到了2013年，IPv6还是不如IPv4流行。你能找到很多办法，采用划分子网的方式，间接地解决有效IP地址不够用的问题。如何让人们在缺乏ISP支持的情况下使用IPv6，或者让ISP在人们还在普遍使用IPv4的情况下提供对IPv6的支持，是一个“先有鸡还是先有

蛋”的问题。人们最初希望手机接入因特网（这是另一个有巨大增长潜力的领域）能推动IPv6跨越发展的临界点。实际上，移动网络内部在越来越多地使用IPv6技术来路由通信流量。这意味着，虽然基于IPv6的基础设施对最终用户还不可见，但已大量应用在了人们看不到的地方。这些持续增加的IPv6设备在静候临界点的到来。

IPv6与装置供电

物联网装置的数量将来会出现爆炸式的增长，我们几乎肯定它们是需要使用IPv6的。但我们也要考虑这些装置的功耗问题。我们能给为数不多的几个装置定期充电，对其做日常维护。我们可能需要时不时地给笔记本电脑、平板电脑、手机、相机和音乐播放器充电。持续地使用电源插座、充电器和线缆虽然有点累人但却可行。但是如果装置的数量非常大，给它们一个个地充电和做维护就非常困难了。因此，这些装置要做到功耗低且非常可靠，同时仍然要能够连接因特网。为了实现上述要求，可能需要把这些装置组织起来，形成网状网络，而这正是6LoWPAN的愿景。6LoWPAN是IETF的一个工作组提出的，在低功耗的无线个域网上使用IPv6的解决方案，使用了诸如IEEE 802.15.4之类的技术。虽然对6LoWPAN及其相关技术的详细讨论超出了本书的范围，我们还是会在第8章介绍嵌入式编程时，讨论一些相关的话题，例如最大化电池寿命等。

IPv6小结

虽然IPv6是（或将是）一个大事件，但本书也将不再对它做更深入的介绍。即便是在2013年，你也能发现，大多数的数据库、大多数的硬件和大多数的开发人员是支持IPv4的（而不是IPv6）。当你正把一个物联网装置从原型转变为产品时，这些IPv4资源是最有用的。即使我们已经离临界点很近了，这样做也是没问题的。因为现有的IPv4服务只需重写少量代码（也可能不需要做任何修改），就能够迁移到IPv6网络。

如果你已经在使用IPv6网络基础设施了，或者你是6LoWPAN早期的采用者，那么你需要了解的IPv6相关专业知识的范围则早已超出本书范围了。

3.3 MAC地址

除了IP地址，每个联网装置还要有一个MAC地址。按我们前面做的比喻，MAC地址相当于信封上面的最终投递地址。MAC地址被用来区分同一物理网络上的不同装置，使得这些装置之间能够交换分组数据。这就涉及TCP/IP协议栈中最低的链路层。虽然MAC地址是全球唯一的，但它们通常只用于一个以太网之内（例如用于一个家庭网络之内）。因此，一个IP分组在被转发的过程中，会被从一个节点转发到另一个节点。当它最终到达某个节点，并且该节点知道接收装置在哪里时，该节点会根据装置的MAC地址把消息传递给接收装置。

MAC即媒体访问控制，是Media Access Control的缩写。MAC地址是一个48位的数字，通常写作6组用冒号分割的十六进制数字，例如：

01:23:45:67:89:ab

对于诸如笔记本电脑之类的大多数装置，它们的MAC地址是烧录在以太网芯片中的。某些芯片，例如Arduino Ethernet板上的WizNet，没有“硬编码”（预先烧录）的MAC地址。这是生产过程的需要：如果芯片是批量生产的，它们当然都是完全一样的。因此，它们无法包含一个互不相同的物理地址。这个物理地址可以保存到芯片的固件中，但这就需要在每一片芯片的固件中包含定制的代码。一个可选的方案是，用一片简单的数据芯片来专门存放MAC地址，然后让WizNet从中读取。很明显，大多数供普通消费者使用的装置均采用了类似的方案，以确保装置每次启动时，使用的都是同一个独一无二的MAC地址。Arduino板作为一个低成本的面向开发者的原型系统设计平台，为了节省时间和成本，没有为了提供一个准确的MAC地址而搞得那么麻烦。Arduino板上有一个小贴纸，MAC地址就印在上面。虽然这种做法看上去有点奇怪，却是有充分理由的：这个MAC地址就是保留给这块板子用的，你用它的话，就可以确保MAC地址的唯一性。在开发阶段，你可以简单地选用一个当前网络中不存在的MAC地址。

WizNet是一家来自于韩国的制造商，专注于生产嵌入式装置上的联网芯片。我们将在第5章看到，很多流行的微控制器都选用了**WizNet**的芯片。

3.4 TCP和UDP端口

在文艺复兴时期的意大利，当信使给富裕的家庭送正式请柬时，他会直接从正门进入。而前来送当季第一批洋葱的食品商人，则从便门进入。这样就可以不妨碍主人的活动，把箱子快速地拿到厨房。下面这幅由John Gilbert创作的版画，其故事题材取自莎士比亚的**罗密欧与朱丽叶**（见图3-2）。这幅画可以提醒我们，凯普莱特家的房子至少还有另一个入口——在朱丽叶的阳台上。如果罗密欧想见他的爱人，这是他唯一的入口。如果他爬上了其他的阳台，或者他进不了门（乳母入睡很快，不会听到他的敲门声），或者他会被朱丽叶愤怒的老爸追得落荒而逃。

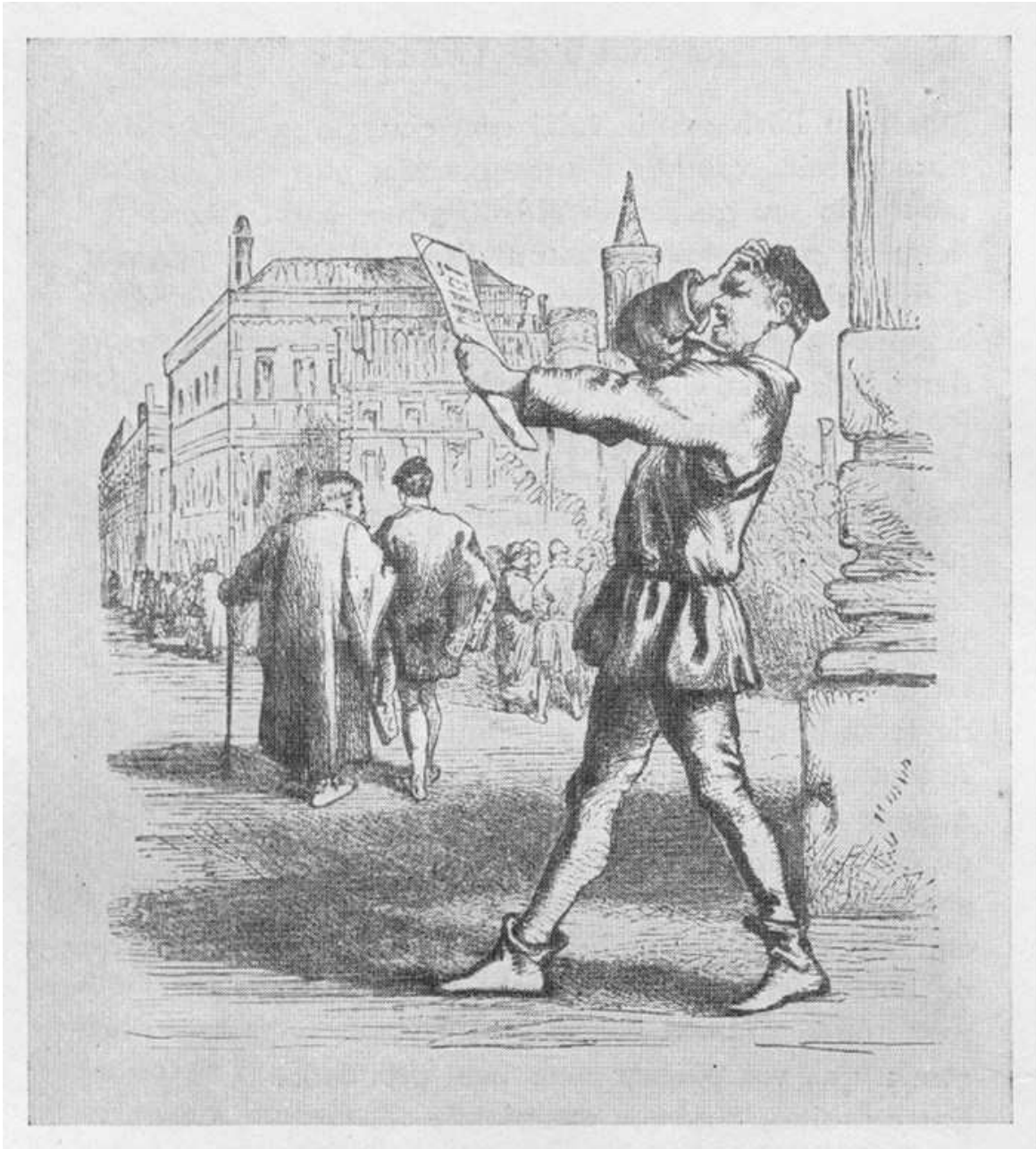


图 3-2 罗密欧与朱丽叶，第一幕，场景2，John Gilbert创作于1873年之前公有领域 http://en.wikipedia.org/wiki/File:Scene_2.jpg

同样，当你在因特网上发送TCP/IP消息时，你必须把消息发送到正确的端口。不同于凯普莱特家的入口，TCP端口是用数字表示的（从0到65 535）。

3.4.1 示例：HTTP端口

如果浏览器想获取一个**HTTP**网页，它通常是把一个**HTTP**请求消息发送到**80**端口。**Web**服务器一直在监听**80**端口，因此能响应这个请求。如果**HTTP**请求消息被发往一个不同的端口，则可能出现以下情形。

- 没有服务在监听这个端口，作为回应，服务器会回复一个**RST**消息（重置**TCP/IP**连接的控制命令）。
- 没有服务在监听这个端口，但防火墙会简单地搁置请求，不会回复。不作响应的目的，是为了阻止攻击者通过扫描每个端口获取服务器的信息（想象一下，罗密欧敲打朱丽叶乳母所在房间的窗户，而她正在熟睡）。
- 客户端判定发送消息到这个端口是不明智的，因此拒绝这样做。谷歌**Chrome**浏览器有一个相当武断的“受限端口”列表，不允许向这些端口发送消息。
- 消息到达了 this 端口，但这个端口准备接收的却不是**HTTP**消息。服务器对这个**HTTP**消息进行解析，把它判定为垃圾信息并断开相关的连接（或者，更糟的行为是，根据这个消息做一些不正常的操作）。

端口**0**～**1023**是“熟知端口”，只有系统进程或系统管理员可以使用这些端口。

端口**1024**～**49151**是“注册端口”，常见的应用程序可以有一个需要注册的常用端口号。不过，大多数服务程序可以绑定这个范围内的任意端口号。

IANA（因特网数字分配机构）负责注册这些范围内的端口号。人们的确可能会滥用这些端口号，特别是在**1024**～**49151**这个范围内的端口。但如果你不清楚自己在做什么，最好还是使用正确的、指派给你的端口号，或者（对于完全定制的应用程序）使用大于**49151**的端口号。

如果主机有多个**Web**服务器程序，你将用到自定义的端口号。例如，在开发阶段，你可能有另一个绑定到**8080**端口的服务器程序：

```
http://www.example.com:8080
```

或者，如果你正在自己的电脑上开发一个网站，你可以使用建站软件内嵌的、用于测试的Web服务器，让它绑定某个空闲的端口。例如，Jekyll（本书网站使用的轻量级博客引擎）有一个运行在4000端口上的测试服务器：

```
http://localhost:4000
```

安全的（加密的）HTTPS协议通常运行在443端口上。因此，以下两个URL是等价的：

```
https://www.example.com  
https://www.example.com:443
```

3.4.2 其他常用端口

虽然你几乎不需要一份用于服务程序的所有端口号的完整目录，但也最好能记住一些日常用得到的常用服务的端口号。例如，你很可能会经常用到下列端口号：

- 80 HTTP
- 8080 HTTP（测试用途）
- 443 HTTPS
- 22 SSH（安全外壳，Secure Shell）
- 23 Telnet

- 25 SMTP （发送电子邮件）
- 110 POP3 （接收电子邮件）
- 220 IMAP （接收电子邮件）

所有这些服务实际上都是应用层协议。

3.5 应用层协议

从底层的有线以太网通信和IP通信，到传输层的TCP，我们已经看到了TCP/IP协议栈中不同层协议的一些例子。现在我们就开始介绍协议栈中最高应用层。做物联网项目的原型设计时，应用层是最有可能与你产生交互的一层（第7章有更详细的介绍）。先稍作停顿，明确一下“协议”的定义，这样做是有益的。

“协议”就是在计算机之间通信时使用的一套规则。这些规则包括怎样初始化会话，消息的格式是什么，等等。协议决定了什么样的输入可以被理解，什么样的输出结果会发送出去。协议也规定了怎样发送消息、验证消息及处理（可能还要纠正）传输错误。

牢记这个定义后，我们就准备好更详细地了解应用层的协议了。先从HTTP开始介绍。

3.5.1 HTTP

因特网不仅仅是Web那么简单，但当涉及物联网时，基于HTTP的Web服务必定会吸引我们大部分的注意力。

HTTP本身是一个简单的协议。客户端请求获得一个资源时，会把一个命令发送到一个URL，同时还会发送一些报头（header）。在下面的例子中，我们使用的是当前的1.1版本的HTTP协议。让我们试着从<http://book.roomofthings.com/hello.txt> 获取一个简单的文档。在浏览器中打开这个URL，就可以看到图3-3所示的结果。



图 3-3 在浏览器中显示Hello World文本

让我们看一下，为了获得这一结果，浏览器到底给服务器发送了什么内容。请求消息的基本结构如下所示：

```
GET /hello.txt HTTP/1.1
Host: book.roomofthings.com
```

我们注意到，消息的内容是纯文本，是用户可读的内容。（这似乎是显而易见的，但并不是所有的协议都这样。例如，在二进制协议中，消息就只被编码为字节形式。）

我们指定使用**GET** 方法，因为我们想要获取网页。（在第7章会对其他方法做更详细的介绍。）然后需要告诉服务器，我们想要的资源

(/hello.txt) , 以及我们正在使用的协议版本号 (HTTP/1.1) 。

接下来的内容是报头, 为请求消息提供一些附加信息。在HTTP 1.1中, **Host** 是唯一必须提供的报头。如果Web服务器在为多个虚拟主机提供服务, 则**Host** 报头可以使服务器把请求消息发送给正确的虚拟主机。

精心编写的客户端, 如你的Web浏览器, 还会传递其他的报头。例如, 我的浏览器能发送如下的请求消息:

```
GET /hello.txt HTTP/1.1
Host: book.roomofthings.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Charset: UTF-8,*;q=0.5
Accept-Encoding: gzip,deflate,sdch
Accept-Language :en-US,en;q=0.8
Cache-Control: max-age=0
Connection: keep-alive
If-Modified-Since: Tue, 21 Aug 2012 21:41:47 GMT
If-None-Match: "8a25e-d-4c7cd7e3d1cc0"
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8)
  AppleWebKit/537.1
(KHTML, like Gecko) Chrome/21.0.1180.77 Safari/537.1
```

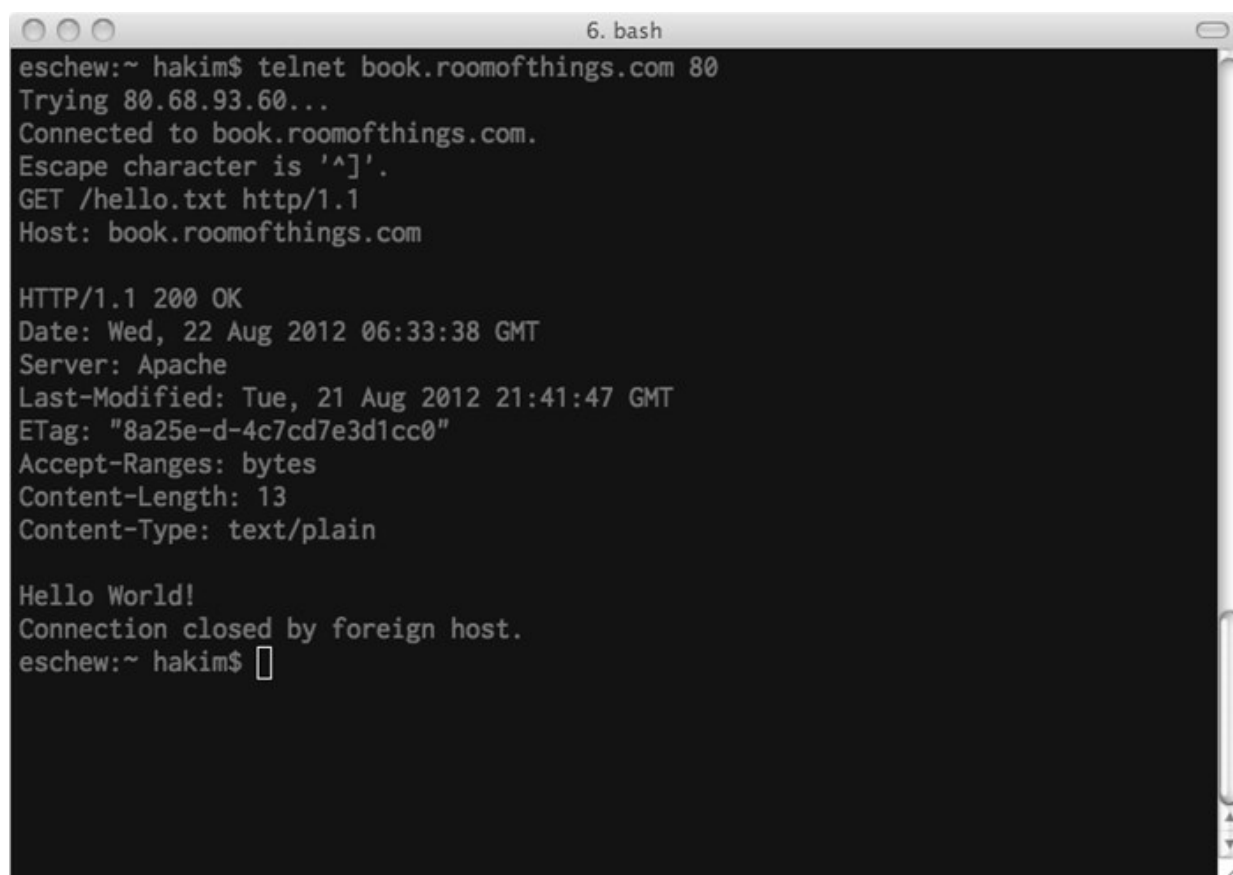
Accept - 报头用来告诉服务器, 当前客户端希望接收哪些类型的内容。它们属于内容协商机制的一部分。例如, 如果我向服务器传递了

```
Accept-Language: it,en-US,en;q=0.8
```

服务器可能会同意向我发送意大利语版本的网页。仅当该页面不存在意大利语的版本时, 才会给我发送英文版的页面。

其他字段用来向服务器提供关于客户端的更多信息（出于统计的需要，或为了绕开已知的程序错误），管理缓存，等等。

最后，服务器会返回其响应消息。我们已经看到响应消息在浏览器中显示的结果。现在让我们再看一下，如果直接使用HTTP协议，完整的请求/响应消息是什么样子。（如图3-4所示。显然，在实际生活中很少需要这么做。即便你在给物联网装置编程，通常也会有现成的代码库可用，可以相对容易地发送请求消息和读取响应消息。）



```
es Chew:~ hakim$ telnet book.roomofthings.com 80
Trying 80.68.93.60...
Connected to book.roomofthings.com.
Escape character is '^]'.
GET /hello.txt http/1.1
Host: book.roomofthings.com

HTTP/1.1 200 OK
Date: Wed, 22 Aug 2012 06:33:38 GMT
Server: Apache
Last-Modified: Tue, 21 Aug 2012 21:41:47 GMT
ETag: "8a25e-d-4c7cd7e3d1cc0"
Accept-Ranges: bytes
Content-Length: 13
Content-Type: text/plain

Hello World!
Connection closed by foreign host.
es Chew:~ hakim$
```

图 3-4 请求/响应消息对

请留意我们是如何使用telnet 命令直接访问80端口的。现在我们可以看到完整的请求消息了。乍看起来，主机名book.roomofthings.com 似乎被我们重复发送了2次。但不要忘了，DNS会把前一个主机名解析为一个IP地址。服务器能看到的全部，就是请求消息，它并不知道发起这个请求的命令是telnet book.roomofthings.com 80。如果域名foo.example.com 也

指向这台主机，Web服务器也许希望能够对 `http://foo.example.com/hello.txt` 有不同的响应。

再看服务器返回的消息。首先是一个状态码**200**（简单地说，它的意思就是“OK”，表示请求消息已被服务器成功处理）。该消息还让我们知道，服务器程序把自己识别为一个**Apache**服务器，内容的类型是 `text/plain`。该消息还包括一些其他信息，用来帮助客户端缓存内容，目的是使将来对网页的访问更有效率。

你可能会奇怪，HTTP消息中怎么没有**超文本**。到目前为止，我们得到的内容还只是纯文本，难道我们不应该使用**HTML**和服务器交流吗？当然不是。**HTML**文档也是文本文档，获取**HTML**文档和获取纯文本文件一样容易。

对服务器而言，无论是回复一个文本文件，还是回复一个**HTML**文档，其处理过程是完全一样的。唯一的区别是，内容类型现在改为 `text/html`。客户端负责读取返回内容中的标记，并以适当的方式显示（如图3-5所示）。

```
6. bash
eschew:~ hakim$ telnet book.roomofthings.com 80
Trying 80.68.93.60...
Connected to book.roomofthings.com.
Escape character is '^]'.
GET /hello.html http/1.1
Host: book.roomofthings.com

HTTP/1.1 200 OK
Date: Wed, 22 Aug 2012 06:42:25 GMT
Server: Apache
Last-Modified: Tue, 21 Aug 2012 21:43:06 GMT
ETag: "8a25f-a6-4c7cd82f28e80"
Accept-Ranges: bytes
Content-Length: 166
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
  <head>
    <title> Hello World! </title>
  </head>
  <body>
    <h1> Hello World! </h1>
  </body>
</html>
Connection closed by foreign host.
eschew:~ hakim$
```

图 3-5 请求/响应消息对（内容类型为HTML）

我们将在本书中陆续介绍更多HTTP的特性，并且它们都是基于简单的请求/响应模式的。在第7章研究Web编程接口（可以认为是基于HTTP的更高层的协议）的同时，还会进一步加深对HTTP协议的理解。

3.5.2 HTTPS：加密的HTTP

我们已经看到请求和响应消息是怎样以简单的文本格式创建的。如果某人对你的连接实施窃听（如果他能接入发送方或接收方的网络，使用诸如Wireshark之类的工具就可以很容易地实施窃听），他就能轻松读取会话内容。实际上，问题并不出在协议格式上。即便会话的内容采用二进制格式，攻击者还是能够通过一个工具程序把会话内容翻译为人类可读的格式。真正的问题在于会话过程没有加密。

HTTPS协议实际上是一个混合体，把标准的HTTP协议建立在了**安全套接字层（SSL）**协议之上。HTTPS服务器监听的是一个不同的端口（通常是**443**），与客户端之间建立的是一个安全、加密的连接（使用某些巧妙、令人着迷的数学技巧，如“**Diffie–Hellman**密钥交换”）。连接建立好之后，通信双方像以前一样，还是使用标准的HTTP通信。

Whitfield Diffie和Martin Hellman于1976发布的Diffie-Hellman密钥交换算法，允许两个人公开交换密钥，还不用担心监听者破译他们随后的会话。算法的实现思路是，通信双方都执行一些数学操作，这些操作实现起来很简单，但对其结果进行破解却十分复杂。例如，求两个素数的乘积是很容易实现的，但如果这两个素数足够大，基于现有的计算能力，攻击者很难通过分解乘积找到这两个素数。通信双方都不会直接发送自己的密钥，而是把密钥与一段共享的信息做运算，把运算结果发送给对方。通过把密钥和这个经由网络发过来的运算结果再做一次运算，通信双方最终得到了一个“共享密钥”。针对这一计算过程，可汗学院提供了一个很好的演示，其所在网页的链接是：

<https://www.khanacademy.org/math/applied-math/cryptography/modern-crypt/v/diffie-hellman-key-exchange--part-2>。

这意味着网络监听者只能识别出请求消息中的IP地址和端口号（在下层的TCP消息中，这两项是公开的信息，因此没有办法对其加密）。发送请求消息时，监听者能看到有数据分组在发送；收到响应消息时，监听者能看到有数据分组返回。监听者所能知道的信息也就这么多了。

3.5.3 其他应用层协议

所有协议的工作方式都大体相似。对于某些协议，其交互过程不仅仅是一个双向的请求和响应。例如，使用SMTP协议发送电子邮件时，你首先需要做的是“HELO握手”，即客户端先愉快地用一个“hello”（SMTP命令都是四个字符长，因此客户端实际说的是“HELO”）介绍自己，然后它会收到一个响应，类似于“250 Hello example.org pleased to meet you!”。不管怎么说，我们都值得花一点时间，用谷歌和维基百科研究一下协议，了解其大体是怎么工作的。你

通常能够找到一个程序库，用来屏蔽通信过程的细节。我们建议你尽可能地使用现有的程序库。糟糕的网络协议实现将给客户端程序和所连接的服务器带来严重的麻烦，可能导致程序错误，或者导致客户端被禁止使用一些有用的服务。因此，你最好使用精心编写过的、经过充分调试且被很多其他开发者使用的协议实现。一般来说，需要你直接实现应用层协议的最符合逻辑的理由是：

- 你所用的平台没有该协议的实现（或者现有的协议实现缺乏效率、不完整或不能用）；
- 出于个人兴趣，你想试着从头开始实现该协议；
- 你正在测试或学习该协议，想让一个特定的请求更容易实现。

3.6 小结

TCP/IP协议集是在因特网上实现数据通信的基础。它的每一层都代表一套通信规则。每一个更高的层都建立在其下各层之上，提供较高层次的抽象。于是，因特网上的大多数开发涉及的是最高层的抽象——应用层，包括为万维网服务提供支持的HTTP协议和将要在第7章介绍的Web编程接口。为了能真正理解通信的过程，做出最佳的技术选择，对下面的各层有所了解是非常重要的。

传输层上的TCP协议提供了很好的可靠性。然而，有时你会认为轻量级的UDP协议更符合你的需求。

网际层的一个重要概念是IP地址。IP地址是访问因特网上装置的关键。但IP地址资源即将耗尽，而移动计算和物联网技术的大规模发展更是加速了这一趋势。于是下一代IP协议IPv6，已在网络基础设施中被大量使用，它将是防止地址资源耗尽的关键。

在最底层的链路层，不是使用IP地址（因为IP地址是在网际层定义的），而是使用MAC地址来识别装置。这一层包括常见类型的局域网，例如有线以太网和WiFi，也包括新的个域网协议，例如802.15.4标准。

在下一章，我们将初步介绍联网装置的原型设计与制作，既包括物品本身，也包括使物品获得魔力的嵌入式硬件和在线软件平台。

第4章 原型设计与制作概述

既然已经了解了若干设计原则和因特网通信的基础知识，我们估计你已经跃跃欲试想创建一个物联网装置了。可能你只想做一个自己专用的装置，也可能你有一个极好的想法，正计划大量生产数以百万计的产品。无论是上述哪种情况，最明智的做法是先做一个装置的原型。

先做原型有很多好处。当创建物联网装置时，你将不可避免地发现设计中的问题，需要对其做出更改或加以持续的改进。和同时修改成百上千的产品相比，改动原型实在是非常省时省力。对于物联网装置，我们通常需要同时构建3个部分：物品本身，使物品具备智能的电路及相关部件，以及所要连接的因特网服务。这其中，对第3个部分的更改相对比较容易。但如果不把每个产品都召回的话，你就没办法改变它们的物理结构和控制器。

因此，制作原型是为了减轻开发的难度，加快开发的速度以及能够对装置进行变更和修改。很多物联网项目开始的时候就是一块微控制器原型开发板，通过连线连接到固定在原型扩展板（如“面包板”）上的各个组件上，并被放置在一个容器中（可能是一个旧铁罐或激光切割的盒子）。这样的原型相对比较廉价，但很有可能得到一个能用但不够精美的物品，而且它的成本和售价可能还会比较高，人们未必愿意买。

在原型设计阶段结束时，你将得到一个能正常工作的物品。对你来说，它可能就够用了。你可以把它展示给朋友看。如果打算做成产品，那么它本身就是一个显而易见的产品，完全可以借助它来让你自己、商业伙伴和投资人相信它的前景是可行的，值得把它做成产品去卖。

最后，利用制造过程来解决量产化和工艺改进等问题。你可以把微控制器原型开发板及相关的连线替换为印制电路板（PCB），后者所搭载的芯片要更为精巧。另外，可以利用工业化的注塑成型工艺来生产数以千计的构件，替换掉那些用3D打印机临时做的塑料构件。最终的产品就会更便宜、更专业化，但对其再做修改也将会付出更多代价。

接下来的几章将详细介绍原型设计与制作的各个方面。

- 第5章：为原型设计选择特定的微控制器
- 第6章：为物品原型设计足够好的外壳和活动部件等
- 第7章：开发物联网装置可以使用的Web服务

在本章中，我们将讨论一些与原型设计平台的选择方法相关的理论问题。虽然我们可以很容易地给出一个简单的回答，例如，“使用 **Arduino**，并且用 **Ruby on Rails** 开发在线服务”，或者“使用树莓派”，等等，但是，通过高高在上的讨论或书本知识来确定最佳的原型设计平台是行不通的。一方面，你的项目有自身的目标 and 需求。另一方面，不断有新型微控制器面世。最佳实现方式和服务器软件包的流行度一直在变化，从而导致快速原型设计技术也不断演进。因此，在做更深入的探讨之前，应该先给出一些通用的建议和背景信息。

4.1 快速搭建原型

在构造原型的时候，你做的第一件事，很可能是用纸和笔快速记录一些想法，或者通过画草图的方式表达一些设计想法。这是一个探索设计思路的重要步骤。我们将对这一步骤中的“画草图”的概念做一下扩展，将其引申为快速搭建软硬件系统。

引申后的“画草图”其实是一个探索问题空间的过程：遍历不同的方案和想法，弄明白哪些可行，哪些不可行。关注的焦点不是原型的保真度，而是尝试实现原型的速度和难易程度。

对于结构设计，这意味着你要把儿时玩的乐高积木玩具翻出来，搭建齿轮结构和三维模型，或者用美工刀对泡沫塑料板或纸板进行裁剪。我们将在第6章对此进行详细介绍。

为了说明怎样快速搭建原型的软硬件系统，请看下面这个例子。

2012年6月，物联网设计公司BERG邀请我（艾德里安）参加了他们举办的首届Little Printer创意日。这次活动聚集了一帮科技迷和创意人士，要求他们在一天内，使用BERG（当时）即将发布的、可以联网的微型打印机，试着做点什么事情。（更多有关Little Printer的信息请见第10章的案例研究。）

大多数参加活动的人都忙着用Little Printer创建新的出版物。他们一般都这么做：通过编写服务器端代码，对数据做各种处理（比如使用Google Calendar API，辨认出飞过上空的流星，等等），再根据处理结果生成图形，最后用狭长的收据打印纸将其显示出来。我觉得，把一种联网装置仅用作系统输出装置，还不够过瘾，因此决定利用这一天的时间再创建一个输入装置的原型。我把该装置命名为**Printernet冰箱**，这算是一种致意，向物联网的一种老创意——物联网冰箱的一种致意。我主要想通过这样一个过程，看一下半自动方式生成的购物清单是什么样子。

根据设计时的各种制约因素（主要受限于Little Printer出版系统的工作方式，也受限于我参加活动时携带的硬件），现在就可以把待解决的

问题分解为三大块：打印出来的出版物的图形设计，便于轻松添加项目到购物清单的物理硬件，将系统各部分联系起来的服务器软件。把问题分解为这三个部分就意味着，至少在开始的时候，可以分别解决每个部分的问题。

第一步是把服务器软件的框架先搭建起来，这样系统的其他部分就可以在此基础上进行开发了。因为这只是个原型，不是一个供成千上万用户使用的系统，我使用了简单的Sinatra框架。Sinatra被用来快速构建Web服务，这和我们将要在第7章介绍的Dancer框架很相似，但Sinatra使用的是Ruby编程语言。

考虑到只会有为数不多的人来观看这个原型的演示，而且也不需要支持多用户，我没有浪费时间添加多用户登录系统，也没有设置任何安全措施用来对服务器端收发的消息进行加密。我们只需要提供足够的基础设施，用来和Little Printer出版系统交互即可，或者等输入装置制作好后，通过添加钩子函数（API hooks）的方式来使用这些基础设施。为了避免在设计细节上花时间，使自己专注于服务器端软件的设计，我创建了一个粗糙的图像，作为出版物图标的预留位置，并快速给它加上了简介：“打印购物清单”。同理，出版物本身是用最简单的静态文本表示的。服务器端一旦收到对出版物的请求消息时，会发送这个静态文本。

到目前为止我们已经可以订阅这个出版物，但只能在Little Printer上打印一行固定的文字。我和参加创意日活动的一名设计师合作，通过对Sinatra应用中构成出版物的HTML、CSS和图像进行微调，对购物清单的外观和版式进行了多次改进。

虽然购物清单的内容还是静态的（你总是得到一个内容固定的清单，例如两瓶牛奶和若干鲜橙汁），但标题图片和文本已经被优化，向用户呈现清单内容的方式也已确定。每一个列表项的项目编号被替换为表示订购数量的数字，并且被移到了每一行的最后。原来显示项目编号的地方放置了一个空的复选框，允许你把在商店里找到的采购项目勾掉。

现在就差实时数据了。生成实时数据的流程需要尽可能简单，添加购物项的过程不能太繁杂。虽然将来的冰箱也许可以替你下单（用户代

理程序将发挥重要作用），但这不是我们现在的目标。我们要实现的只是一个工具，而不是一个可以自主做决策的装置。

因为我的包里只有一块**Arduino Ethernet**板，硬件平台就只好选它了。如果把这个**Arduino**板用作产品的一部分，以太网连线会增加安装的复杂性，并且加密数据的收发会超出其处理能力的极限。但对于一个快速原型来说，**Arduino**板便于在不同电路之间连线，可以轻松变更其上的代码，即便从长远的角度考虑也是一个利大于弊的选择。

把一个基本的按键连到**Arduino**板之后，就可以编写一些简单的代码。每当按键被按下时，就可以触发这些代码的执行。第一步先实现通过串口输出一些文本，可以在我的笔记本电脑上立刻看到这些文本（电脑通过**USB**线与**Arduino**板相连）。

既然基本的物理交互已经实现，下一步就是让**Arduino**板和**Web**服务建立连接，这需要对二者的软件进行修改。我为**Sinatra**应用添加了一个新的**API**调用，用来向列表中添加新项目，并且用**Web**浏览器对其进行了测试（如果什么地方出错，用浏览器可以更容易地查看错误）。

对于一个更复杂的项目，我会在**Rails**和**Sinatra**之间选择前者作为**Web**框架。这样我就可以使用**RailsAdmin**模块做一些检查，例如查看项目是否被正确地添加到了数据库中。此类辅助模块使你能专注于快速实现一个功能。在你还不确定是否保留这一功能前，无需分心去构建大量额外的基础设施。在本例中，我可以轻松地把静态购物清单切换为使用实时数据。使用实时数据，既是一项被开发的功能，也是一种调试手段。

接下来，我找来一段用来从**Arduino**上发送**Web**请求的范例代码，对其做了修改，使其能够调用服务器端新编写的**API**函数。完成这些代码后，现在只要按一个按键，“牛奶”就会被添加到购物清单中。成功了！

下一步是添加更多按键，因为对于购物清单应用程序而言，只能订购牛奶并不实用。于是，另外两个用来订购“奶酪”和“橙汁”的按键被添加到了原型中。

在进行了一轮快速的用户测试（就本例而言，就是向创意日活动的其他参与者展示一下）之后，当前设计的一个问题凸显出来。虽然把一个新选择的项目记录到服务器上费时不超过1秒，但如果是快速地连续按下各个按键，这个处理时间还是显得过长，导致后续按键的按压不被记录。

考虑到当天所剩时间不多，并且我可以在演示的时候避开这个问题，我没有把按键相关的用户接口代码从网络通信代码中分离出来，因为这需要编写大量的代码。作为权宜之计，一个表示“忙”的LED灯被添加到了面包板上。每当Arduino进行网络通信时，它都会点亮。

忙完硬件和软件部分后，已经没有什么时间用来给输入装置做一个外壳了，当然也是因为我的包里再也装不下任何结构材料了。我便临时用便签纸做了一个按键说明，这至少可以让用户界面看上去一目了然。便签纸上还有剩余的空间，因此我对进一步扩展原型的方式做了一个提示，把“添加条码扫描器”作为一条备注写到了按键说明的下面。这是对装置实际工作时的情况做深入思考后的成果。你也许需要若干按键，以及可以对这些按键对应的无包装食品集合进行重新配置的Web界面，而所有的带包装的食品则用条码扫描器来扫描输入。



图 4-1 完成后的Printrun冰箱原型

4.2 熟悉程度

选择原型设计平台时，熟悉程度是另一个需要考虑的因素。例如，如果你已经是Python编程能手了，那么你可能会选择诸如树莓派之类的平台。这样的话，你就可以用你熟悉的语言编写代码，好过从头开始学习Arduino。

对服务器端软件的选择，显然也是同样的道理。在创建Printernet冰箱原型之前，艾德里安没有用过Sinatra。之所以会选择Sinatra，一是因为他需要一个简单的Web框架，二是因为他以前写过几个Ruby on Rails应用程序，对Ruby已经比较熟悉。

如果你已经能熟练的用一片一片的泡沫塑料板构建三维结构，我们也不建议你为了学习激光切割技术而忽略这项技能。

4.3 成本与开发难度

虽然从原型系统开发的难易程度来看，选择熟悉的平台很具吸引力，但一个平台的成本（原型和批量生产的成本）和该平台所需的开发工作量之间的关系也是值得考虑的。这种权衡存在一定的灵活性，但你最好能选择一个和最终产品在性能/处理能力方面处在同一等级的原型平台。这样你在项目后期就不太可能对项目成本，甚至是批量销售的可行性感到意外。

例如，当前创建电子装置的最廉价方案可能是选用**AVR**微控制器芯片，你可以从供应商那里以大约3英镑的价格买到这种芯片。但这只是芯片的价格，你还需要弄明白怎么把芯片的各个引脚与其他组件相连，怎么把新代码写入这个芯片上。对很多人来说，这不是可行的制作初始原型的平台。

如果把预算提高到20英镑左右，你可以考虑**Arduino**或与之类似的平台。虽然还是使用**AVR**微控制器芯片，但这个芯片已经集成在了一块单板之上。单板上面的引脚标签能使你可以轻松地快速连线到外围组件。单板上配备**USB**接口，可以用来连接计算机。此类平台拥有良好支持的集成开发环境（**IDE**），可以使编程过程更轻松。不过，出于性能和内存使用效率方面的考虑，你仍然需要使用**C++**编程。

再把预算提高到30英镑左右，你可以考虑**BeagleBone**开发板。**BeagleBone**可以运行**Linux**，拥有足够的处理能力和足够大的**RAM**，因此能够使用更高级的编程语言——**JavaScript**。在基于**JavaScript**的并发编程工具包**Node.js**中，已经有现成的库，用来对板上的输入/输出引脚直接进行操控。

如果不想选用嵌入式平台，可以考虑使用智能手机。智能手机可能要花费300英镑左右。虽然智能手机和嵌入式平台看似是完全不同的东西，但它们有很多相同的功能（相对廉价的嵌入式平台也因此更有吸引力）：联网功能（通常是通过**WiFi**或**3G**网络，而不是以太网），输入能力（是指触摸屏、按键、摄像头，而不是电子元器件），输出能力（声音、屏幕显示、振动）。有各种各样的高级或低级的编程语言

可以为手机编程，例如Objective-C、Java、Python、HTML和JavaScript等。

普通的PC机也是构造原型时的一个选择。这些PC机的价值从100英镑到1000英镑不等，具备丰富的连网能力和I/O扩展能力。你可以使用任何熟悉的语言为它们编程。更重要的是，你很可能手头就有一台现成的PC机了。

对于第一个原型来说，成本可能不是最重要的。如果你手头已经有智能手机或PC机，那么用它们来做就很方便，这实际上是一个零成本的选择。虽然用一台通用计算设备构建物品原型的做法看似有点不上道，但这要看具体的情况。为了验证想法是否可行，或者引发人们对项目的兴趣以让其提供协作或资助，这样做是完全恰当的。

在这个阶段，用最容易的方式实现原型应该说是最明智的。现阶段选用你可以负担得起的最强大的平台是合乎情理的。

当然，如果你的装置需要与外部实体交互（吹泡泡、转动钟表的指针，或者从刻度盘获取输入值），你会发现PC机没有对此类需求进行优化。PC机上没有暴露在外的GPIO引脚（虽然以前有人用PC机的并口临时性地解决了这个问题）。毫无疑问，电子系统的原型开发板更适合做此类工作。我们稍后将介绍怎么把PC机和原型开发板两者的优势结合起来。

硬件和编程方面的选择取决于你所掌握的技能，这是一个重要的考虑因素。所以，我们谈到原型开发的难易程度时，一个明显的疑问是，这是对谁而言的难易程度？

对很多刚刚接触硬件开发的人来说，Arduino工具包是一种非常好的选择。输入输出引脚的选择很简单，你只需要具备理解接线图的能力，当然最好还懂一点电子学方面的基本知识。从编程的角度看，交互过程实质很简单——从GPIO引脚读取数值或把数值写到GPIO引脚。Arduino的编程语言是C++，虽然现在（21世纪初）很少有人会认为它是初学者的最佳之选，但Arduino工具包已经把调用过程抽象为`setup()`函数和`loop()`函数。更重要的是，IDE把编译后的代码下载到装置之后，程序会自动进入运行状态，直到你把装置的供电断

掉。虽然Arduino开发板的性能有限，但它也因此存在一个优点，即与Arduino开发板的交互过程是高效流畅的。

比较一下Arduino板和计算机。如果你已经知道怎样用C#、Python或JavaScript开发应用程序，那你就有了一个好的开端。但如果没有这些基础，你需要首先评估和选择一种编程语言，然后设法弄明白怎样编写和运行程序，让程序自动运行起来。严格来说，相对于和外表有点怪异的电路板打交道，这些事情都比较容易做，也更容易理解。不过，选择的自由也增加了其自身的复杂性。

另一种选择就是把通用计算设备和微控制器结合起来使用。微控制器用来连接刻度盘、LED灯和电机等低级的组件，计算机或智能手机用来做复杂的处理工作。诸如Arduino之类的开发板可以很容易地通过USB接口连接到计算机。你可以使用任意编程语言，通过标准的串口操作和开发板进行通信。

一些智能手机也有处理复杂事务的能力。但手机和Arduino都属于“装置”，理论上不能作为主设备对Arduino进行控制。（连接手机USB接口的另外一方负责做主设备。）对此问题，有一个有趣的解决方案，即在Arduino板上安装一块USB主设备**盾板**，这样Arduino就可以充当USB主设备的角色，理论上就可以对手机进行控制了。Android配件开发工具（ADK）就是采用的这种方案。在现实生活中，手机也的确能够做复杂的处理，与因特网通信等。

在原型平台的选择上，从来没有唯一正确的答案，你总是要做各种折中和取舍。但不要因此拖延你开始制作原型的步伐，因为在原型平台的选择上，确实没有“错误答案”。原型是一个让你开始行动的物品。如果想搞清楚哪个平台是装置的最佳选择，那么你在制作原型过程中学到的这方面的东西，将比从任何一本书（包括本书）中学到的都要多。

案例研究：Bubblino

让我们详细了解下Bubblino，看一下在实际的项目中原型设计的过程是怎样的。

项目伊始，**Bubblino**就选择了**Arduino**硬件平台，部分的原因是：该项目的初衷就是想展示一下怎样用**Arduino**制作物联网装置。因此，最初的硬件方案就是把**Arduino**板和一个电机相连，成为一个买来就能用的泡泡机。最初的原型使用的是支持蓝牙功能的**Arduino**板，目的是能够通过蓝牙连接艾德里安的诺基亚手机。该手机使用的是**Series 60**平台，可以用**Python**对其编程。手机联网后，能够向**Arduino**板发送一个表示新推文数量的数字。**Bubblino**收到这个数字后，会将其视作以秒为单位的时长，会在此时间内吹泡泡。

虽然**Python**是一个脚本语言，以简单易学著称，但在当时，**Series 60**平台尚不成熟，实现过程比预想的要艰难。因此在制作第二代**Bubblino**时，艾德里安转而使用他笔记本电脑上的**Perl**脚本，但工作流程没有变化，还是向**Arduino**发送一个数字，其实现方式就是打开对应于蓝牙连接的**COM**端口并对端口执行写操作。如果使用当前基本的只带**USB**接口的**Arduino**板，发送数字的实现方式还是一样的。这种实现方式造成了一种限制，即不管是一代还是二代的**Bubblino**，它们必须通过蓝牙或**USB**线连接主设备后才能工作。

虽然对于一个用于演示的样机来说，让驱动装置的笔记本电脑一直开着，这种做法非常合理，但为了实现产品化，**Bubblino**已经做了相关的改进：你现在已经可以委托我们制做一台**Bubblino**了，几周之内就能完成。虽然**Bubblino**不是批量生产的产品，但为了能节省成本与时间，还是有必要对设计方案和工作流程做些调整。所以，现在的**Bubblino**选用的是**Arduino Ethernet**板。这意味着**Twitter**搜索和**XML**处理都是在**Bubblino**上完成的，因此只要连接以太网，**Bubblino**可以完全独立于计算机工作。

在使用**Perl**脚本的第二代**Bubblino**中，可以使用全功能的**XML**解析器。在**Perl**软件库**CPAN**（任何流行的现代编程语言都有类似的库）中，有很多可供选用的**XML**解析器。对于**C++**语言，当然也存在类似的解析器。然而，由于内存大小的限制，在**Arduino**上加载和处理大段的**XML**数据并不恰当。当用**Bubblino**下载**Twitter**搜索结果的**Atom**提要时，**Bubblino**只是简单地对返回的**XML**数据进行扫描，从中搜寻<published> 标签以及其中的日期字段。通

常情况下，我们有理由建议开发者不要通过简单地扫描关键字或通过匹配正则表达式的方式去解析XML数据，但对于资源受限的装置，这样做是可以接受的。

最后，Bubblino的概念被应用到了iPhone应用程序“Bubblino和他的朋友们”中。这个程序的功能是用关键字对Twitter进行搜索，根据所选对象的不同，产生动画效果或者声音效果。可选对象除了泡泡机外，还可以是它的某位朋友（程序中引入了新的角色，例如海盗鹦鹉等）。

4.4 原型和产品

虽然在项目起步阶段，原型开发的难易程度是一个主要的考虑因素，或许还是阻碍项目启动的最大障碍，但之后在原型的基础上构建更多装置（可能成千上万）的过程则会带来一系列全新的挑战和问题。

4.4.1 修改嵌入式平台

进入量产阶段，出于控制成本或尺寸的考虑，你很有可能会考虑迁移到不同的平台。如果你最初使用的是一个没太多约束的、功能强大的编程平台，你会发现，把代码移植到一个限制较多、价格便宜、尺寸更小的装置会带来很多挑战。你应当意识到这个问题。如果第一个原型是建立在PC机、iPhone、BeagleBone或其他任何有助于获得投资和合作伙伴的平台之上，你就要准备着手把各种能吸引人的功能移植到最终的目标平台上。

当然，如果在原型开发阶段使用的是一个资源受限的平台，你可能不得不对代码做一些限制。在Arduino提供的2KB内存中进行动态内存分配可能不是很有效率，因此你应该不会考虑使用字符串或复杂的数据结构。如果你迁移到了一个更强大的平台，你可以用更现代和更高级的方式重写你的代码，或者只是简单地利用一下更快的处理器速度和更多的RAM。但你要留意，新平台拥有的I/O功能是否有变化？同时，你还要考虑学习新技术和新语言所需要投入的时间。

实际上，你常常会发现，你并不需要改变平台。你可能会考虑把Arduino原型开发板替换为AVR芯片（Arduino用的就是这个芯片）和各种实际需要的元器件，并在一块定制的印制电路板上把它们连接起来。我们将在第10章更详细地讨论这个话题。

4.4.2 原型结构和批量个性化定制

原型结构所采用的制造技术很可能无法直接应用到批量生产过程。虽然制造技术可能会改变，例如，用注射成形代替3D打印，但多数情况下，这不会导致结构本身的变化。

数字制造工具能允许每个构件稍有不同，这是一个有趣的方面，能让你在某种方式上对每一个装置进行定制。有可能进入产品生产阶段后，你仍然需要继续以单件的形式制造某些可改变的部件。而批量个性化定制，正如这个名称所揭示的那样，意味着你能够提供独一无二的产品，同时也有可能为此收取额外的费用。

4.4.3 迁移到云端

和装置本身相比，服务器端软件是最容易实现从原型到产品的转变过程的。我们之前提到过，这个过程包括从一个基础的Web框架迁移到某个更复杂的Web框架（具备添加用户账户之类的功能）。不管你之前选用的是哪种编程语言，你都能够找到一个采用同种语言的Web框架。这意味着大多数的业务逻辑在迁移过程中都不会有大的改变。

另外，在以前，在转入产品阶段后，你需要购置一台更强大的服务器。现如今，如果服务运行在云计算平台之上（例如亚马逊AWS平台），你可以根据需求的变化对服务进行动态扩展和收缩。

案例研究：利物浦DoES

之所以要考虑制造技术，可能只是想得到一个更美观的装置——哪怕只是为了自用。在后续章节中，我们会看到一些用来给物联网装置原型制作外壳的方法。但毫无疑问，从裸露在外的电路板，到用乐高积木或可重复利用的盒子做外壳的物品，再到用专门定做（激光切割或3D打印）或批量加工的盒子做外壳的装置，这或许也是一种改进。

在我们的办公室和利物浦DoES创客空间，中央加热系统被接入了因特网。这套被称为YAHMS的系统，其构成部分包括：在办公室内外安装的用于测量温度的一定数量的传感器，控制加热器开关的执行器，用于管理定时器和提供Web接口的服务器软件。和很多没有接入因特网的加热系统一样，该系统有一套基于定时器的程序，用来自动地确保基本的舒适度。不过这个基本的温度调节机制是可以被取代的，用户可以登录YAHMS网站，查询当前温度，并决定是否打开或关闭加热装置。因为系统是基于Web的，

不管你此时是坐在办公室中，还是在寒冷冬天的某个周六正准备从家里出发去上班，这套系统都一样好用。

温度传感器就是一些裸露在外的Arduino板，John负责管理它们，这也是他承担的项目。连接到锅炉的线缆安装整洁，电子元器件都被隐藏了起来。人们每天用到的界面被设计成了最简洁的形式，无论是用桌面浏览器还是用智能手机，其使用效果都一样好。与硬件部分定位于发烧友级别的使用者的情况类似，软件部分的一些功能当前还没有可以用来操作的用户界面，需要使用数据库操作命令进行调整。显然，这套系统目前还不适合批量销售，但它已经能用了，其最常用的功能已经做了良好的封装，并且该系统也和DoES捣鼓一切有趣技术的大氛围相符合。

另一个出自于DoES的案例是DoorBot。它最初的构成就是一台联网的PC机，其平板显示器通过一个位置适当的窗口面向走廊放置。DoorBot被用作信息终端装置，可以显示网络摄像头拍摄的办公室场景和即将发生的事件列表（通常获取自Google Calendar），还能向任何来宾显示一条问候消息。目前，它唯一支持的输入装置是RFID读卡器，DoES的会员可以用自己的RFID卡（Oyster、Walrus和DoES会员卡等）进行登记。DoorBot也连接了一个扬声器，用来在会员出入的时候播放个性化的音调或消息。开发这个装置就和在计算机上运行软件一样简单，其难点主要包括：在工作时间之外如何打开和关闭屏幕显示，以及当电力和网络断开或恢复时，应该怎样应对，等等。鉴于这个装置提供的功能与PC机相似，考虑PC机之外的方案似乎有点不太正常。但如果需要对这个装置进行升级，使其能够覆盖多扇门，或者要向其他公司兜售这个方案，我们就能立刻做出新的权衡了。

对于有的办公室，在门口附近放置一个PC机机箱不一定很合适。利用诸如iMac、笔记本电脑或平板电脑之类的一体化计算设备可以解决这个问题，但这些设备要比原先用的普通PC机贵很多（如果有一台很少使用的PC机，实际上可以认为它是“免费”的，因为它本来就放在那里没人用）。诸如树莓派之类的小型嵌入式计算机可能是理想的选择，因为其成本相对低廉，能运行Linux，并且有HDMI输出接口。

我们的同事John和Ben，最终把DoorBot移植到了树莓派上，这是一次很宝贵的学习经验。尽管从理论上讲，树莓派和运行Linux的PC机具有相似的功能，这个移植过程只是一个小的改变，但还是需要投入不少时间。花时间做这件事的主要动力是，我们想把DoorBot扩展到另外两个房间，即一共需要三套装置。这样就需要找到一个更小型化、更廉价和更优雅的方案。我们将在下一章中更详细的介绍这个移植过程。

对于那些不是明显适合使用PC机且需要连接电子元器件的项目，你更有可能考虑使用各种性能等级的微控制器：从基础的Arduino及其类似产品，再到诸如BeagleBone之类性能更高的微控制器。

如果你的原型装置成功构建在了一个小型且廉价的平台之上，那么，如果突然需要批量生产该装置，则这个装置的成本已经比较合理了。你已经证明了自己的构想能够在一个廉价、小尺寸和资源受限的单板上实现。当然，如果装置对硬件的使用已经达到了硬件能力的上限，那么你会发现，增加任何新功能都意味着不得不马上对硬件升级，升级到所选芯片的一个功能更强的版本，甚至需要升级到一个不同的平台。如果装置已经上市销售了，此类硬件上的改变会带来一个不便之处：将现有装置的固件升级到一个新版本变得不那么容易了。

如果你开始就选择了一个性能比较强大的平台，你就同时获得了各种便利条件，例如，能够比较自由地进行动态内存分配，可以使用由封装良好的API构成的库，等等。但如果你不得不把代码移植到一个资源更有限的装置上，那么这些便利条件就变成了让你感到痛苦的不能实现的假设了，你可能需要重新编写全部的代码。当然，重写代码未必是一个问题，但你需要把它计入到开发成本之中。

4.5 开源与闭源

如果你很较真的话，你可能要花费一生的时间去论证开源和闭源的定义，实际上一些人确实以此为职业。概括的说，我们打算讨论的是这两个问题：

- 作为创造者，你对自己的知识产权的主张
- 用户自由修改成果的权利

你也许是一名技术多面手、发明家、程序员或设计师，总之我们认为本书的很多读者都有一定创造才能。作为一名创意人士，你内心可能非常纠结，一方面你非常渴望学习物品的工作原理，想对其进行修改或重复利用，另一方面，你又担心其他人在你的设计、发明或软件上做类似的工作，这样你有可能得不到预期的认可和收益。

实际上，在开源和闭源方式之间进行选择是一个相当有趣的话题，特别是把它们应用于像物联网装置这样的软硬件兼有的综合系统时。虽然很多人可能已经下定决心选择了其中的一条道路，我们还是建议你至少考虑一下在项目中同时使用这两种方式的可能性。

开源硬件协会

在物联网出现的同时，还出现了一个类似的（或许层次稍显更高）的现象——创客运动的兴起。亲自动手把玩技术的理念和策略，鼓舞了充满热情的业余爱好者和专业人员，使他们能够立刻拿起电烙铁和工具箱，开始创造各种新事物：从机器人和3D打印机，到交互式的艺术作品和电子游戏。

创客运动和创意空间（**hackspace**）有着很大的共通之处，其主体都是由志趣相投的人构成的人际网，这些人通过互相帮助的方式学习新技能，用新奇的方式尝试使用各种技术。创意空间和开源软件社区所具有的民主与开放的工作方式，都融入到了创客文化之中，使得共享设计和代码成为了这些组织与社区所一致默认的观点。

随着这个社区的不断发展与成熟，这种共享就聚合为开源硬件这种更正规的形式。

2010年3月，开源硬件研讨会的与会者们开始讨论并制定开源硬件的定义。在接下来的11个月中，通过在线交流以及在第一届开源硬件峰会上的讨论，这个定义被完善并提炼为开源硬件

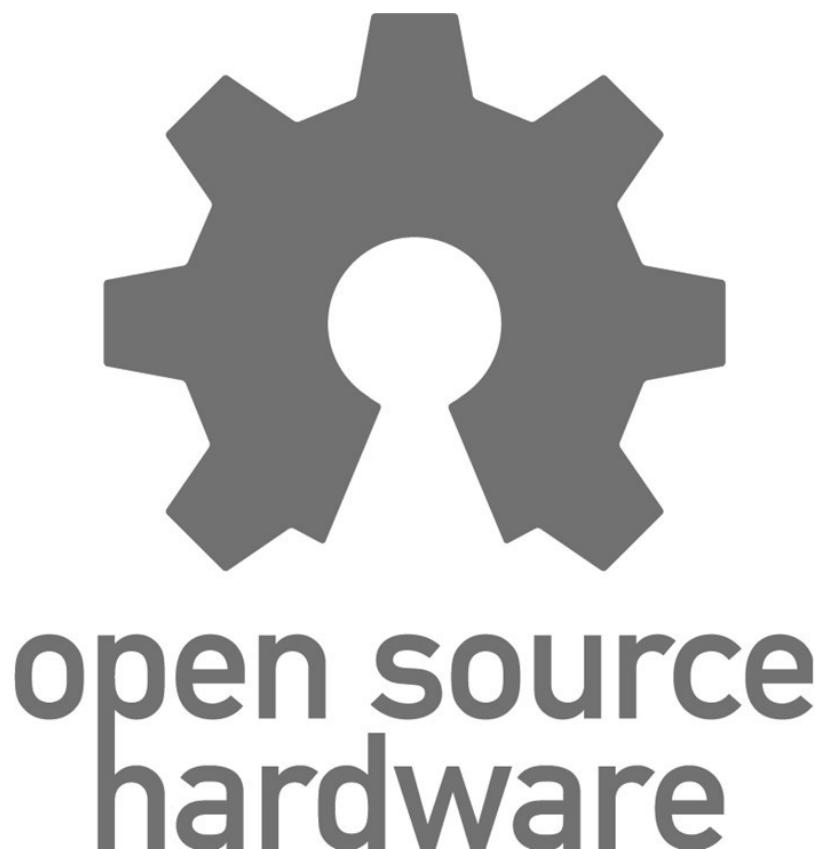
（OSHW）的原则与定义声明的1.0版

（<http://freedomdefined.org/OSHW>），并于2011年2月发布。相关的工作目前还在进行中，写作本书时，1.1版本的文档还在草拟中。

开源硬件的标识现在经常伴随其定义一同出现。设计师们可以在他们的产品上使用这个标识，用来表明产品的源文件是可获得的，可以出于学习和再利用的目的使用这些文件，也可以基于这些文件做扩展，设计出新的产品。

开源硬件峰会已经成为每年举办一届的盛会，大量的黑客、创客、开发者和设计师在每年9月都聚集在一起，讨论和颂扬开源硬件。

2012年6月，开源硬件协会（<http://www.oshwa.org>）正式成立。在我们写作本书时，该组织还在探索自身的定位，在如何支持开源硬件峰会，以及就开源硬件相关事务为公众提供教育支持等目标上，正在拟定具体的方案。符合开源硬件相关条款的产品可以使用开源硬件的标识。



开源硬件的标识

<http://oshwlogo.com>

4.5.1 为何选择闭源

维护知识产权往往是默认的做法，尤其是对于规模较大的公司来说。对于某些源代码或某项设计，如果你声明拥有其版权，那么其他人想把相同项目推向市场时，仅仅参照你的操作指南是做不出来的。他必须要对各种软硬件功能进行逆向工程处理才成。另外，一味模仿别人的设计也会侵犯版权。你可以分别借助商标和专利来保护视觉设计和软硬件中的独特元素。

尽管借助良好的法律支持来解决问题显得费时费力，但较大的公司还是会选择这条途径来尽量保护自身所取得的专利并行使相应权利。如果你是在这样的公司中开发物联网装置，在企业文化的范畴内做事确实会容易些，只不过在想要尝试不同的解决途径时，你需要去说服管理层、市场部门和法务部门。

如果是自己单干或是在一个小公司中，你可能只需注册一个独特的商标，借助版权来保护你的成果。需要说明的是，在项目开始时选择闭源方式，并不妨碍你以后以开源形式发布项目成果（但当你以开源形式发布成果后，就不能再简单地取消许可了）。

你可能非常想保护自己的知识产权（特别是在你和亲人的生活依赖你的创意这种情况下），这是完全可以理解的。但要注意，对于这些权利能对养家糊口有多大的实际帮助，而开源又能带来什么好处，你需要好好权衡一下。

4.5.2 为何选择开源

在开源体系中，你面向全世界创建项目和发布源文件。你可以把软件代码发布到GitHub（<http://github.com>），把电子设备的原理图发布到Fritzing（<http://fritzing.org>）或SolderPad（<http://solderpad.com>），把装置外壳的设计图发布到Thingiverse（<http://www.thingiverse.com>）。

如果你还不习惯这些做法，也许会觉得这样做不太正常：为什么要把自己很在乎并且费了很大劲才完成的东西免费分享给别人？也许以下这些理由能说服你：

- 你可以从喜欢你工作成果的人那里获得积极的评价；
- 这是对你工作成果的一个公开展示，既能提高你的声誉，又能为你带来新的机遇；
- 使用你工作成果的人能给出建议，实现新功能，或修正错误；
- 让人们尽早对你的项目产生兴趣，获得支持和广泛关注。这是一个花钱也买不到的好处。

当然这也是一种礼物经济：你也可以在自己的项目中使用他人以开源的形式免费贡献的内容。因特网上有各种的论坛和在线交流频道，人们可以在其中比较自由地讨论他们的项目，这样做有助于他们获得上述的各种好处。

如果你有点技痒，想通过某个项目施展一下才华，用开源形式发布这个项目无疑是最好选择。你会从喜欢你的设计（包括关于此设计的博客文章）的人那里得到鼓励。在你的项目面临困局时，这些鼓励是无价的，使你能够坚持下去。如果某人用你想不到的方式修正了一个错误，这可能会为你节省数小时不愉快的调试时间。如果足够幸运的话，你可能还会被人称为“那个玩泡泡机的家伙”，因而一举成名，或许还会被邀请到各种会议上去谈论你的LED电路。

如果你做的是是一个很正规的项目，你可能还是会发现，至少对你的一部分工作而言，选择开源是一个正确的决定。

开源的缺点

开源似乎有一个明显的缺点：“人们会偷走我的创意！”但实际上，问题没你想的那么严重。通常，当你向别人介绍自己的创意时，对方很难听得进去，因为他们也在等着向你介绍他们的好创意呢（自私的家伙）。如果人们的确使用了你贡献的开源内容，他们很可能是以自身所喜欢的方式去使用这些内容。这说明创意的空间其实还是非常大的。

不过，以开源形式发布工作成果可能会占用更多的资源。正所谓鞋匠的孩子没鞋穿。你在为其他人做设计时，不得不采用比较高的标准，而如果只是为自己做设计，则往往会忍不住去走捷径。当你做好一个能运转的原型后，你可以稍微庆祝一下。然后你就需要花费大量时间和资源，返回头去对原型进行修改和完善，这样你才能够以比较体面的形式发布你的工作成果。

当然，解决这个问题的正确做法是在项目最开始的时候，就把所有的工作成果即时放到开放的存储空间中，以公开的方式做开发。这样才更符合“开源的方式”。你可能会需要一些时间去适应这种做法，但这样做是可行的。

在你以开源的形式发表工作成果之后，你可能仍然有义务对其进行维护 and 提供支持，或者至少能通过电子邮件、论坛和聊天室等方式来回复相关问题。尽管可能没有付费用户，但你可能还是愿意维护由用户形成的这个社区。如果你的确是自愿做这些事情，并为此投入了时间，你确实可以选择何时减少或停止投入，这是你的自由。但是，在

成功建立一个社区前就放弃支持的项目不能被归为一个成功的开源项目。

做一位好公民

开源项目的正确做法是什么？这是一个值得我们思考的问题。做开源方面的工作，你需要具备一定程度的声望，开发者们会因此被吸引到你的项目中来。如果你正在追求这方面的声誉，你要确信自己值得被人信任。如果你先是声称自己做了一个开放平台，然后数月之后才发布了几个库，而且没有文档或文档质量很差，那么这会被认为是一种无礼的做法。此外，开源项目应该尽量支持其他开放平台。例如，对于驱动程序库来说，把项目的平台限定为你所控制的装置，这自然无可厚非，但对于其他宣称开源的项目而言，这种假定是有问题的。

在某种程度上，做一位好公民会抵消掉礼物经济模式的优势。但任何一种经济学模式都有各自的一套公民规则，这也是很自然的事情。

把开源作为竞争优势

虽然好公民云集的开源社区与礼物经济模式都很让你为之感动，但开源也有可能成为一种竞争优势，这一点也很重要。

首先，如果你想得到一个被很多人测试、改进和调试过的软件，则使用开源产品通常可以被认为是没有风险的选择。只要该产品没有使用极端“病毒化”的许可（例如AGPL），你确实没有理由不使用它——哪怕是在一个闭源项目中。毫无疑问，你可以用元器件从头开始构建自己的微控制器，编写自己的用于电机控制的库、自己的HTTP协议栈和Web框架。或者你也可以使用开源产品，例如，**Arduino**，**Arduino**自带的**servo**库和以太网协议栈，以及**Ruby on Rails**。以上的开源产品都存在与之功能对应的商业软件，如果你选用了商业软件，除了要考虑成本因素外，你也将不得不依赖于某个公司的支持论坛（这跟在网上有着大量资源的开源产品完全不同）。

其次，积极地使用开源产品能使你的产品有机会获得广泛关注。正如你在本章中看到的那样，本书会经常提及**Arduino**。其实，**Arduino**不是最强大的平台，并且肯定能做进一步改进，这一点可以轻易得到证明。**Arduino**的成功，除了得益于它的低成本，更重要是因为其极高的

关注度。由于Arduino的设计是开源的，很多其他的公司都在生产兼容Arduino的单板，以及诸如盾板之类的兼容组件。这也导致了一些有趣的现象，例如Arduino引脚布局缺陷

（http://forum.arduino.cc/index.php/topic,22737.0.html#subject_171839）。实际上这是一个设计错误，但却被其他有着相同用户群的制造商们原样复制了。

如果一个开源项目做得足够好并快速获得了良好口碑，那它就很容易受到热情的追捧，成为一个平台。极客社区在选择产品时，通常不会选择一个商业的“黑匣子”，他们更多会选择那些具有开放的Linux命令行接口，或能通过XML之类的开放协议与之通信的产品。这种社区能成为你最大的盟友。

把开源作为战略武器

积极使用开源产品还有更深层次的目的，即通过策略性地使用开源产品来提升自己的利益（同时削弱竞争对手的优势）。

在“让互补品低价

化”（<http://www.joelonsoftware.com/articles/StrategyLetterV.html>）一文中，软件企业家Joel Spolsky¹表明了这样一个观点：很多公司在开源项目上有大笔投资，他们这么做的目的其实就是想让互补品低价化。在经济学领域，互补品就是指和你的产品有配套关系、需要同时购买的那些产品和服务，例如，DVD和DVD播放机就是互为互补品。

¹ 被誉为“程序员部落酋长”的Joel Spolsky是著名网站Stack Overflow创始人，他的博客Joel On Software以谈论软件开发及管理问题闻名，2卷博客文集的中文版已出版《软件随想录》，另1卷也即将由人民邮电出版社出版。——编者注

如果两种商品互为互补品并且其中之一的价格发生下降，则对这两种商品的需求很可能都会增加。因此，很多公司把对互补品的开源版本进行改进作为一种手段，以此来提升对自家产品的需求。例如，对于制造微控制器芯片的厂商，对运行于自家微控制器芯片之上的开源软件框架进行改进，将有助于售出更多芯片。

云计算领域的思想领袖Simon Wardley这样写道：

对很多人而言，“开源”这个词让他们想到的是嬉皮士们的理想主义——热爱自由的极客们免费把自己的工作成果分享给他人。同样会有很多人认为这是他们能做的最反资本主义的事情。这些人和特洛伊人一样，太容易上当了。

——<http://blog.gardeviance.org/2012/04/be-wary-of-geeks-bearing-gifts.html>

尽管把核心业务开源确实有些风险，但如果能试着把其他业务开源，而这些业务又恰好是竞争对手的核心业务，那么这样做将有助于削弱竞争对手的实力。因此，谷歌公司开源发布Android，削弱了苹果公司iOS平台的影响力。Facebook开源发布了Open Compute，以有效帮助大型数据中心的维护，削弱了谷歌公司在这方面的竞争优势。

Facebook显然需要高效的数据中心。通过开源相关代码，该公司有机会从很多聪明的开源程序员那里获取更多的智慧与创意。但Facebook却没有将社交图谱的核心算法进行开源。

这些动态的相互作用在物联网中显得更为有趣，若干不同层面的事物，包括结构设计、电子元器件、微控制器、与因特网的信息交换、后端API和应用程序等，通过彼此之间的交互形成最终的产品。这也是很多人试图成为中间件层领导者的一个原因，Xively就是其中之一（Xively对开发者免费，虽然目前没有开源，但很多非核心功能是开放的）。

在做原型开发时，这些都是相对次要的考虑因素，但要对这些事情有所了解，因为这有助于你理解其中存在的风险和机遇。

4.5.3 混合使用开源和闭源

我们已经讨论过，可以把你的很多库开源，而让核心的业务保持闭源。虽然很多企业只在开源和闭源之间选择其一，但不要低估两者共存的可能性。只要不罔顾事实，否认自己使用了开源软件，你还是有可能成为一个开源世界的“好公民”的。你可以在获取开源项目诸多好处的同时，通过贡献自己的工作成果或在论坛里帮助他人来为开源项目做贡献。

虽然我们这两个作者都热衷于开源的理念，但实际上，我们的工作成果并不都是开源的。我们承接了一些商务工作，而客户有保留知识产权的要求。还有一些工作，因为不够完善，不值得为其做额外投入，所以没有被开源发布。

艾德里安的Bubblino项目混合使用了开源和闭源许可模式：

- Arduino代码是开源的；
- 可以获取原理图，但没有为此做特别的宣传；
- 服务器端代码是闭源的。

服务器端的部分代码没有开源，因为物联网装置的一些配置细节可能关系到商业利益。

4.5.4 在大众市场项目中选择闭源

在以下这种极端情况下，必须要选择闭源许可：你能确切地预见到，某个项目不仅能获得成功，而且还会有巨大市场，即可以成为大众市场商品。在你需要借助用户口碑发展一个平台的时候，开源用户社区是一个很好的盟友。但如果你能让一条现有的供应链和分销链为你服务，因而能让产品第一个上市并且成本低廉，则没有比获得这些有利因素更重要的事了。

让我们看一下智能温控器Nest。智能的能量计量与控制是一个很多人都在尝试开拓的领域。一旦某个国际电力公司决定向其所有用户推出电力监测装置，此类项目就会立刻成为大众市场项目。因此，此类项目很容易被别人模仿和抄袭。例如，对于某个技术水平高，各项配套条件都准备就绪的制造商，它很有可能这么做。如果还能获得原理图和全部源代码，对产品进行逆向工程破解所需的花费都可以省掉了。

对于一个物理装置而言，把它转变为可批量生产的产品需要付出巨大的成本和努力。这显示了供应链的重要性，它能影响项目在其他方面的选择。2001年，Paul Graham²满怀热情地说明了选择正确的编程语言（对他而言就是指Lisp）的重要性，因为他的竞争者们选用了开发速度慢得多的其他编程语言，所以他能在竞争中胜出

(<http://www.paulgraham.com/avg.html>)。当然，就把产品推向市场的速度而言，选择什么开发平台不是关键因素。开源和闭源之间的对立状况对产品上市速度也有一定的影响。

2 硅谷创业教父，著有《黑客与画家》。——编者注

4.6 利用社区资源

我们在前面的章节中提到了社区，不过还是要说明一下，把社区自诩为开源项目的独有特征是不诚实的。

在你考虑选择哪一个平台的时候，有没有社区资源可以利用是一个非常重要的考虑因素，它的存在至少会对你有所帮助。这也是我们当前选择支持**Arduino**平台的一个主要原因。如果你对某个组件或库存有疑问，或者有一个问题（例如怎样用电位计旋钮控制一个伺服电机），只需要在谷歌搜索中用关键词“**arduino servo potentiometer**”（**arduino 伺服 电位计**）搜索一下，就能找到相关的**YouTube**视频、博客文章或代码。

其他很多不错的平台，如**Chumby Hacker**板，确实也有自己的发烧友社区，但其规模会比较小。如果你在做一件鲜为人知的事情，并且需要获得详细的技术援助，要想找到一个曾做过同样事情的人可能会很困难。

随着项目的发展壮大，受关注程度也可能会变得重要——例如，如果想雇用能在你所选择的平台上做事情的人。对于人数不多且目标明确的团队来说，这个问题可能不太重要，因为他们可能已经在新的、不太知名的平台上积累了很多专业技能。但这还是一个需要考虑到的问题。

当你还是一名缺乏经验的创客时，选用一个能有他人为你提供指导的平台是非常有价值的。如果你能参加本地创客的聚会，如利物浦创客之夜（**Maker Night Liverpool**），或者在世界各地的创意空间举办的类似活动，你也经常能找到愿意手把手教你使用**Arduino**或其他类似系统的人。这个人可能是这方面的专家，也可能只是在上一次聚会时学过一些入门知识（让**LED**灯闪烁，或者用压电扬声器播放歌曲**Mary had a little lamb**）。这些聚会对学生和导师来讲都非常有用。

参加本地的聚会也是讨论自己的项目或从其他人那里取经的最佳途径。虽然讨论你的项目在某种程度上是在开源该项目，但你可以控制要说的内容和对象。如果你还没有做好开源的思想准备，当面交流的

方式能让你不太胆怯，因为那要比把自己的聪明想法直接发布到网上感觉好些。

在与其他人分享想法和实施方案，或者与其讨论问题时，你可能会面临这样一种窘境：在众目睽睽之下，自己可能会表现得像个白痴。在许多网上社区的部分板块中，人们都对这种担心报以同情和理解，他们的宽容程度往往会超出你的预料。但尽管如此，因特网的匿名性还是会使得藏在“面具”后面的人们未必会像你期望的那样友善，能对你提供足够的帮助——他们有时甚至还会做出粗鲁的回应。一般来说，如果是初涉物联网创客社区，在创意空间中进行面对面的聚会是一种让你觉得更加友好有益的方式。

与本地或网上的创客社区保持接触的原因之一是，用交互设计师和BERG硬件工程师Andy Huntington的话说，我们现在处于**Geocities of things**阶段，即物联网的开拓阶段，就像**Geocities**曾经处于制作网站和博客的前沿位置一样。的确，某些物品的设计可能还很笨重，某些物品也可能毫无意义，很多人只是在重复别人的工作。但是，这些源源不断的创新终将激发出下一代的成功业务和改变世界的项目。现在是投身物联网热潮的好时机。

4.7 小结

现在，对于怎样构建你的第一个物联网原型装置，你已经有了相当不错的基础，对许多问题已经有了大体上的了解了。

做原型开发时，你一方面需要构建一个能让你对所做的项目有更多了解的装置，另一方面还要留意，如果你的想法以后能得到印证，到时候怎样把原型转变为产品。在这两者之间，总是要做些权衡。

下面的几章将依次介绍物联网装置的三个主要的组成部分：嵌入式计算和电子装置，物品本身的物理结构，以及与装置相交互的因特网服务。对于怎样着手对这些组成部分进行原型开发，我们将届时予以详细介绍。

不管你是因为**Ruby on Rails**是开源的，所以决定用它设计服务器软件，还是因为你是一个有经验的**Python**程序员从而选择使用树莓派开发板，又或者为了避免从头开始因而选择从**Thingiverse**下载某个关键组件的**3D**设计，在本章介绍的这些概念的指导下，你都将能够构造出每一个组件。

第5章 嵌入式装置的原型开发

试想，你要做一个装置，并且知道它应该包含某种交互或电子相关的部分。如果要把想法转变为一个真实存在的物品，第一步你要做什么？

你可能会尝试使用若干不同的零部件，对它们的性能一一进行验证。对于你的第一个原型，这是一个好的开始。当你在因特网上查询过类似的项目，或者浏览过组件零售商，例如RS（www.rs-components.com/）或者Farnell（www.farnell.com/）的产品目录后，你将得到一个可能会用到的组件和模块清单。这个清单将助你实现自己的目标。

在电子技术和微控制器方面涉猎越多，你手头闲置的以及从以前的项目中剩余下来的零部件就越多。当坐下来尝试某些想法时，你可能已经翻寻过收集的各种零部件，找到了和打算用的组件功能相近的组件，或者你已购买了一套新组件。通常情况下，这两种方式都会用到。

初次尝试自己的想法时，选择容易实现的平台是典型的做法。这样做可能是因为你已经熟悉了这个平台，另外也可能是因为这有助于控制原型开发的成本。即便你知道正在使用的单板不是最理想的选择，但如果这样做能更快、更节省地尝试实现某些功能，那么你的选择在当前阶段也是正确的。

在原型开发时，选择使用手机、笔记本电脑和台式机等计算能力过剩、表面看起来更加昂贵的设备开发最初的软件，有时也是正确的选择。如果有现成的手机或计算机可用，用它来开发原型实际上不再是昂贵的选择。

然而，如果你还没有玩过什么电子装置，也没有若干闲置不用的开发板，那么该购买哪一种呢？本章将针对几种比较热门的选择，对它们的一些功能和差别进行介绍。这些所谓的热门的选择，会随时间变化。但你应该能搞明白，怎样把上一章讨论过的标准应用到所有正在考虑选用的单板上。

本章的开始部分会介绍一点电子学方面的知识。因为不管你最终选择了哪个平台，你需要构建和连接的其他电路装置差不多都是一样的。之后我们介绍了四种不同的平台，你可以把它们作为物联网原型装置的基础部分。这四种平台不是唯一的选择，但这些都很有代表性。在本章结束的时候，你将能更好地权衡不同的选择，也将对作为实例的几种单板有足够的了解，可以选择其一做进一步的探索了。

5.1 电子电路基础

在专心研究微控制器和嵌入式单板计算机的各种细节之前，让我们先把注意力集中到可能会用到的电子元器件上。

如果对学习使用焊接技术心里发怵的话，请不要为此担心。你在构建最初的原型时不大可能用到焊接技术。大多数的原型制作工作可以在所谓的无焊面包板上完成。无焊面包板使你能够以“插入即可”的方式把各种元器件连接起来形成电路。这也意味着你能快速、轻松地尝试不同的可选方案。

谈到电子元器件，我们最好把它们分为两大类来研究。

- **传感器**：装置借助传感器来获取信息，感知周围环境中的事物。
- **执行器**：执行器是装置的输出部件，包括电机、灯等，使装置能在外部世界中做一些事情。

上述两类电子元器件都可以通过多种方式与计算机进行通信。

最简单的方式是采用数字I/O接口，该接口只有两种状态，用来表示按钮是否按下，或者LED灯是否可以点亮等。这些状态通常经由通用I/O端口（GPIO）的引脚与处理器的内部状态建立联系，处理器中的数字0被映射为接口电路中的0V电压，数字1被映射为一个置位电压（通常是处理器的工作电压，一般是5V或3.3V）。

如果除了on/off两个状态之外，你还想分辨更多的状态，就需要使用模拟信号。例如，如果你连接了一个电位计，用来读取一个旋钮的位置，你得到的是一个和旋钮的当前位置相关的不断变化的电压。同样，如果不希望电机只有停止和全速运行两个状态，想让它以适当的速度运转，你需要给它提供一个介于0V和最大额定电压之间的电压。

因为计算机是纯粹的数字设备，所以你需要用一种方法将现实世界中的模拟电压值转换为计算机中的数字值。

模数转换器（ADC）能够测量不断变化的电压。微控制器通常会在其内部集成若干个ADC。这些ADC能把介于0V和一个预定义的最大值（通常是5V或3.3V，和处理器的电压相同，有时是一个固定值，如1V）之间的电压值依据ADC的精度转换为一个整数。Arduino集成了10比特的ADC，其缺省的测量范围是0V到5V。0V电压对应的转换结果是0，5V电压对应的输出是1023（10比特内存空间能存储的最大值），而介于0V到5V之间的电压值对应的读数大小则与电压值大小有关。1V对应的读数是205，而512这个读数对应的电压是2.5V，以此类推。

数模转换器（DAC）实现的功能正好和ADC相反。DAC允许根据数值的不同生成大小可变的电压，但它一般不属于微控制器的基本功能。这是因为，一种被称为脉冲宽度调制（PWM）的技术，能通过快速改变数字信号的on/off状态，使输出电压的平均值符合你的期望，这样就能提供和DAC近似的功能。和DAC相比，用来实现PWM的电路更简单。对于某些应用，如调整LED灯亮度，PWM其实是首选。

复杂一些的传感器和模块具备诸如SPI总线和I²C总线之类的接口。这些标准化的机制使得这些模块能够和外界通信，因此传感器、以太网模块或SD卡能够借助此类接口连接到微控制器。

当然，我们没办法对所有的传感器和执行器进行一一介绍。下面我们将介绍一些最常见的类型，让读者对技术方面的可能性有一个了解。

5.1.1 传感器

按钮和开关可能算是最简单的传感器了，用户使用它们对输入状态进行控制。电位计（旋转式和直线滑动式）和旋转编码器则能测量位置的改变。

对环境量进行测量也是比较容易的。光敏电阻（LDR）能对环境光的强度进行测量，热敏电阻和其他类型的温度传感器则能让你知道当前温度。测量湿度和水分含量的传感器很容易构建。

麦克风显然是用来监测声音变化的，但某些类型的麦克风使用的压电元件也能用来测量振动。

距离感测模块是利用从某个物体反射回来的红外或超声波信号进行测距的，此类模块很容易买到，并且其连接方式也和连接电位计一样简单。

5.1.2 执行器

灯是一个最简单最有用的执行器，其对应的电路很容易创建，并且能产生明显的输出效果。发光二极管（LED）通常发红光和绿光，但也有白色和其他颜色可选。RGB LED的设置过程有些复杂，但允许把不同亮度的红绿蓝光混合起来，产生任何颜色的光。还有一些更复杂的可视化输出方式，例如用LCD屏幕显示文本或简单的图形。

压电元件除了能对振动做出反应外，也能用来产生振动。因此，可以用压电式蜂鸣器生成简单的声音和音乐。或者，也可以把输出连接到扬声器，合成更复杂的声音。

很多情况下，你可能还需要使用某种器件在真实环境中移动物品。电磁阀可以被用来产生一次快速的推送动作，例如把一个球推离壁架，或者轻轻敲击物体表面发出乐声。

电机是更复杂的执行器。顾名思义，步进电机以“步”为单位改变位置，通常前进一个固定的步数就能使其旋转一周。直流电机在运转的时候，只是依照指定的速度改变位置。这两种类型的电机都可以单向或双向旋转。另外，如果希望电机能转到一个指定的角度，需要使用伺服电机。虽然使用伺服电机能更容易地控制其输出，但它的转动范围比较小，通常不超过180°（而步进电机和直流电机的转动角度则没有限制）。无论是采用上述哪一种类型的电机，你通常是想把它连接到某个装置，以改变其移动范围或把旋转运动转变为直线运动等。

如果想了解更多有关计算机或微控制器接口方式的内容，浏览一下Arduino Playground上的“硬件接口”页面

（<http://playground.arduino.cc/Main/InterfacingWithHardware>）是一个不错的开端。虽然这个页面中的内容主要是针对Arduino的，但其中大多数的建议都可以在稍作变通后用于其他平台。如果想对电子技术有更深入的了解，推荐阅读*Electronics For Dummies*一书。

5.1.3 原型电路的演进路线

从电子技术的角度看，原型制作的起始点通常是“面包板”。可以通过把元器件和连接线插到面包板上的方式构造电路，无需任何焊接，这就使得实验过程变得简单。当你对面板上的实验结果感到满意时，通常会把各个元器件焊接到某个万用板上，这样就能保证稳固持久的电路连接，也能防止连接线插错地方。

在万用板电路的基础上做进一步改进，往往意味着要学习PCB布线了。这项工作没有听上去那么复杂，至少对于简单的电路而言是这样的，主要是学习使用一个新软件和理解一些新术语。

对于小批量生产的情况，很可能会使用通孔元件。此类元器件的引脚会穿过PCB上的小孔，并且往往会采用手工焊接的方式。通常我们把自己的设计实现为一块与既有的微控制器平台配套的单板。在Arduino社区中，该单板一般被称为**盾板**。这种方式让你能够依靠自己的力量做出产品，不用担心需要从头开始设计整个系统。

电路板之旅

让我们看一下部分Bubblino电路从最初的测试电路、到万用板电路，再到PCB成品的演进过程。

(1) 创建电路的第一步通常是在面包板上构建电路。这样就能够确定元器件布局的过程中，很容易地做出调整。见图5-1。

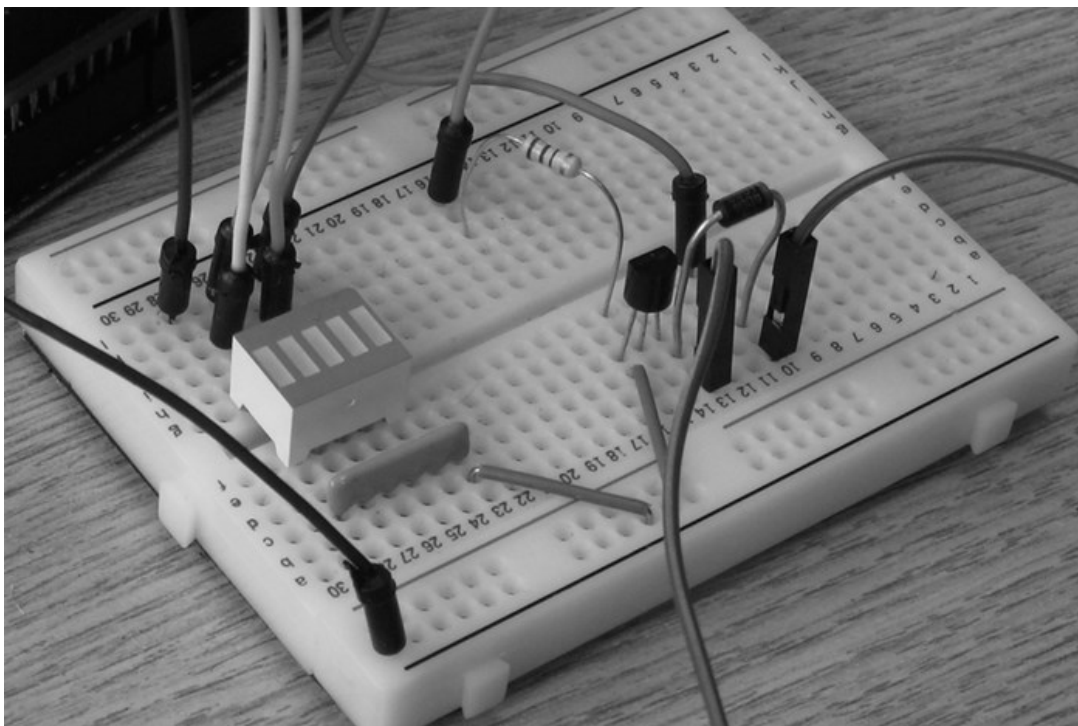


图 5-1 面包板

(2) 对面包板电路的工作状况感到满意后，把这个电路焊接到万用板上，使电路布局得以固定。这意味着，你将不再需要担心某一根连线会固定不牢。如果只打算制作一个这样的电路，这样就可以了，无需再做改进。见图5-2。

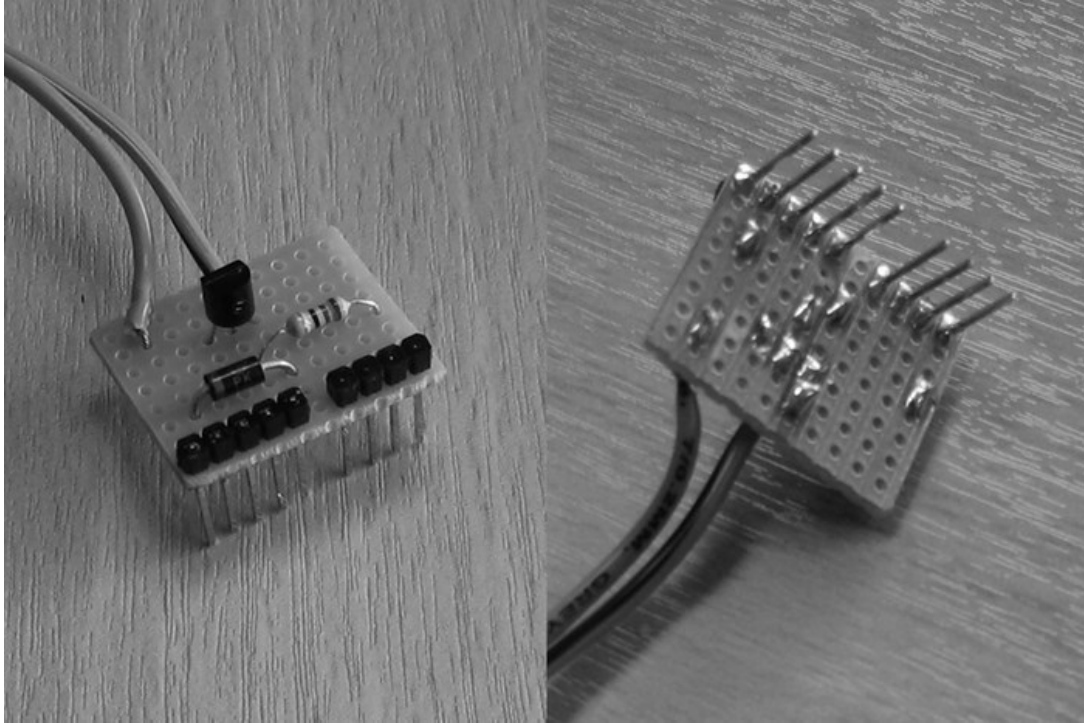


图 5-2 万用板

(3) 如果需要制做很多个这样的电路，或者想做得更专业一些，可以把电路做到PCB上。在PCB上构建电路会更加容易，因为每个元器件的位置都已被标出，只有在安装元器件的位置才会有孔洞。短路的可能性也会减小，因为元器件之间的走线会被阻焊区域隔开。见图5-3。



图 5-3 PCB

当想对电路部分做更进一步的改进时，采用多层电路板可以做到移除微控制器板上不需要的元器件，并且能转而使用表面贴装元件——芯片的引脚被焊接到芯片所在的一侧，从而便于使用自动化生产线组装单板。

第10章将详细介绍PCB的设计，以及制造过程中的各种选择。

5.2 嵌入式计算基础

本章的后续部分将对几个不同的嵌入式计算平台进行研究。在此之前，有必要先介绍一些后面将陆续用到的概念和术语。

介绍一些背景知识非常重要，因为很多读者可能对于微控制器是什么没有多少概念。虽然我们一直在说，处理器正变得越来越便宜，也越来越强大，但你也不能只是把一堆PC组件弄到一块儿，就称为物联网产品。如果你曾经打开过一台PC机，你应该已经看到，它是由若干可以分开的模块组成，每个模块都提供了不同的功能。这些模块包括：主板和处理器，用作RAM的一两块小电路板，以及提供长期存储能力的硬盘。总之，计算机由很多组件构成，这些组件能提供各种通用的功能，并占据一定的物理空间。

5.2.1 微控制器

物联网装置采用了更紧密集成、微型化的解决方案——从最基础级别的微控制器，到功能强大的片上系统（SoC，System-on-Chip）模块。这些系统把处理器、RAM和存储器都组合到了一个单独的芯片上，这意味着它们更适用于特定的领域，比PC机上提供类似功能的组件更小巧，并且也更容易融入到用户的设计中。

这些微控制器是无数传感器和工厂自动化设备的引擎。尽管32位甚至更多位的计算已经出现很久了，但它们是8位计算在这个世界上的最后的堡垒。微控制器是一种能力非常受限的器件，因此，尽管32位微控制器的价格已经大幅下滑，开始挤压8位微控制器的市场空间，仍然有人愿意选用8位微控制器。通常，微控制器只提供几千字节的RAM空间和几万字节的存储空间，尽管在能力方面有各种限制，还是能被用来完成很多事情。

如果8位计算和以KB为计量单位的RAM让你觉得恍惚瞬间回到了20世纪80年代，想起了Commodore 64 或Sinclair ZX Spectrum等早期的家用计算机，这是可以理解的。8位微控制器和这些早期的计算机内部工作机制相同，内存容量也大体相当。但微控制器经过了这么多年的发展，还是有所改进。和20世纪80年代的同类产品相比，现代的微控制

器芯片体积小，能耗低，并且运行速度要比20世纪80年代的同类产品快5倍左右。

与台式机处理器市场被两家主要的制造商（Intel和AMD）垄断的情况不同，微控制器市场有很多的制造商。可以和汽车市场的情况做个比较。在汽车行业，有许多不同的汽车制造商，每个厂商生产用途不同的一系列车型。同样，在微控制器市场，也有很多制造商（Atmel、Microchip、NXP、Texas Instruments等），每个厂商生产针对不同应用的一系列芯片。

随处可见的Arduino平台基于Atmel的AVR ATmega系列微控制器芯片。Arduino板上包含了各种GPIO引脚和ADC电路，能方便地与各种传感器、指示灯和电机进行连接。因为采用这些微控制器的装置只专注于完成一项任务，因此它们不需要使用操作系统的大多数功能。和基于SoC或PC机的解决方案比，其代码更简单，占用空间更小。

在此类系统中，需要较多资源的功能通常是由额外的单一功能的芯片提供。有时，这些芯片要比控制它们工作的微控制器芯片功能更加强大。例如，对于Arduino Ethernet板用到的WizNet以太网芯片，其RAM空间是Arduino板自身RAM的8倍。

5.2.2 片上系统

定位介于低端的微控制器和功能全面的PC机之间的是SoC（例如，BeagleBone和树莓派）。与微控制器类似，这些SoC在一个单独的芯片中整合了处理器和若干外围器件，但它们通常要比微控制器具备更多功能。SoC通常会使用主频在几百兆赫兹以上的处理器，其高端解决方案甚至会使用主频在GHz级别的处理器。SoC上集成的RAM也不会只有几千字节，而是在兆字节的数量级。SoC上也往往不再集成存储单元，而是普遍采用SD卡作为存储单元。

SoC具备更多的能力，这意味着它们需要某种操作系统来管理和分配资源。可以选择的嵌入式操作系统有很多，包括闭源和开源两类。它们或者来自于专门的嵌入式系统提供商，或者来自于大型操作系统厂商，如微软、Linux厂商和社区等。随着处理器变得越来越廉价，Linux等流行度和熟悉度都比较高的操作系统正被日益广泛采用。

5.2.3 选择平台

怎样为物联网装置选择正确的平台？回答这个问题的难度，和搞清楚生活的意义差不多。不是说这个问题无法回答，而是因为答案的数量几乎和可能存在的设备数量一样多。你在选择平台时，会综合考虑价格、性能和功能等各种因素，以便于达成自己的目标。即便你已经选定了一个解决方案，也不意味着别人在解决相同问题时不会选择一套完全不同的方案。

我们现在就开始选择一个用于原型制作的平台。本节的后续部分将要讨论的是，在你决定如何构建装置时，需要考虑的因素——这些因素之间可能会相互制约。

我们介绍的这些需要你做的决策，对本章后续部分和第10章介绍的内容都是适用的。

处理器速度

处理器速度，确切说是处理器的时钟频率，描述了处理器能以多快的速度处理正在运行的程序中每一条单独的机器码形式的指令。处理器速度越快，处理器执行起指令来就越快。

时钟频率虽是用来衡量原始计算能力的最简单指标，但不是唯一的指标。你也可以根据各平台的数据手册和规格说明文件中提供的数据，用每秒百万指令（MIPS）做比较。

某些处理器缺少对浮点数计算的硬件支持。如果代码中包含很多复杂的数学运算，速度相对较慢但硬件上支持浮点数计算的处理器，要比性能指标稍好但硬件上不支持浮点数计算的处理器处理代码的速度更快。

通常，对功能类似的系统做比较时，可以把处理器速度作为需要权衡的若干因素之一。微控制器的时钟频率往往是在几十兆赫兹的数量级，而SoC的运行速度能达到几百兆赫兹，甚至能达到几吉赫兹的水平。

如果项目不涉及非常复杂的处理工作，例如，只是需要有联网能力和很基本的感测能力，则某些类型的微控制器就能满足需要。如果装置将要处理很多的数据，例如，对视频进行实时处理，那么你就需要考虑选一个SoC平台了。

RAM

RAM是系统的“工作内存”。RAM越大，你能做的事情越多，在选择代码的算法时灵活性越大。如果你需要在装置上处理大的数据集，这将决定你需要多大的RAM空间。你经常能找到一些办法，通过调整代码（参见第8章的内容）或把处理过程迁移到在线服务中（参见第7章的内容），以迂回的方式解决内存不够用的问题。

对于多大的RAM空间比较合适这个问题，很难给出一个确切的选择标准，因为不同的项目有不同的标准。但不管怎样，不大可能选用那些RAM少于1KB的微控制器。如果想运行标准的加密协议，至少需要4KB或者更多的RAM。

对于SoC板，特别是当你打算在上面运行Linux操作系统时，我们建议至少要有256MB的RAM。

连网

对物联网产品而言，怎样把装置接入网络是一个关键的考虑因素。使用有线以太网常常是最简单（通常能即插即用）、最便宜的接入方式，但需要用到网线。无线解决方案显然能避免使用网线，但配置过程却较为复杂。

WiFi部署广泛，是一个现成的接入网络的基础设施。但和它的一些竞争者相比，WiFi接入的成本相对较高，在功耗方面的优化程度也较低。

和WiFi相比，其他短距离无线接入方式在功耗和成本方面有一定优势，但这通常是以减小带宽为代价的。ZigBee就是一项这样的技术，是特别针对传感器网络和智能家居之类的应用场景设计的。新近出现的蓝牙低功耗协议（对应于蓝牙4.0版本）支持非常低的功耗配置，这一点和ZigBee类似。但蓝牙低功耗协议可以被植入到手机和笔记本电

脑上的标准蓝牙芯片中，因此该协议将会更为迅速地被人们所接受。当然，现存的低版本的蓝牙标准是另一种可能选择。在低端市场，还有一些长期存在的标准可供选择，例如**RFM12B**。**RFM12B**的工作频段在**434MHz**附近，而前面提到的其他几种选择使用的是**2.4GHz**附近的频段。

对于部署在偏远地区或者户外的装置，使用移动电话网络是最好的选择。对于低带宽且能允许较长时延的通信需求，可以采用**SMS**之类的基本通信方式；如果对数据传输率有较高要求，则要使用和智能手机一样的数据连接，例如**3G**网络等。

USB

如果在装置附近有一台功能更强劲的计算机，通过**USB**接口连接到计算机，就能轻松获得供电和连网能力。你在购买微控制器时，可以选择支持**USB**接口的版本，这样你在构建外围电路时就能少用一个芯片。

除了以设备的形式存在，一些微控制器还能承担**USB**主机的角色。这种配置方式使你能够把一些通常用于连接计算机的物品连接到微控制器上。例如，可以使用**Android ADK**套件连接电话之类的装置，可以连接附加的存储装置和**WiFi**适配器等。

诸如**WiFi**适配器之类的装置，通常要依赖于主机系统中额外的软件，如网络协议栈。因此，此类装置更适合用于**SoC**之类的功能接近于计算机的平台。

功耗

运行速度比较快的处理器通常比相对较慢的处理器更费电。对于便携式的需要使用电池供电的装置，或者需要依赖于某种非常规供电方式（如太阳能）的装置，功耗是一个需要重点考虑的问题。即便能方便地连接到供电设施，也要考虑功耗问题，因为低功耗可能是一个值得拥有的特性。

不过，处理器可能会支持睡眠模式。在此模式下，功耗会非常低。这样你就可以用一个速度较快的处理器快速地执行操作，随即转入低功耗

耗的睡眠模式。因此，即便是在一个低功耗嵌入式装置中，使用一个功能比较强大的处理器也不一定是坏事。

与传感器和其他电路的接口

除了能接入因特网，你的装置还需要做一些其他的交互：用传感器收集环境相关的数据，或提供到电机、LED灯和屏幕等的输出。你可以通过SPI和I²C等外围总线连接到这些外围电路，通过ADC或DAC模块读取或写入可变的电压值，或者通过通用的GPIO引脚实现数字量的输入输出。不同的微控制器或SoC解决方案提供了不同数量的接口组合。

物理尺寸和外形

随着芯片制造技术的持续改进，决定芯片尺寸的，早已不再是在硅晶片上构造电路时所有晶体管和其他元件占据的空间。现如今，芯片的尺寸受限于用来连接PCB上的外围元器件的引脚数量。

如果PCB采用的是传统的通孔式的设计（常见于手工自制的电路板），芯片的引脚间距通常设定为0.1in。即便芯片与外围电路之间只需要很少的连接，例如有16个引脚，这个芯片的周长最终还是不能小于约4cm。对于比较复杂的芯片，其需要的引脚数量很容易就能超过100个，在PCB上为周长约25cm的芯片留出足够的空间可能还是有点难度的。

采用表面贴装技术能减小引脚的间距，因为这样就不再需要为了连接引脚而在电路板上钻孔。把表面贴装技术和在芯片背面设置引脚的设计技巧相结合，就有可能使复杂芯片的尺寸得到有效控制（不会需要和桌面一样大的PCB）。

每个引脚的尺寸能减小到什么程度，这要取决于制造过程的能力和公差。对于一些表面贴装设计，引脚的尺寸做的足够大，可以手工焊接到自制的PCB上。对于其他的表面贴装设计，需要使用专业生产的PCB，并且要用高精度的上下料机把PCB定位到正确的位置。

因为总是要在芯片尺寸和装配过程的复杂性之间做出权衡，很多芯片都被设计为多种可选的外形尺寸，即采用不同的封装形式。这就允许电路设计者针对特定的应用，选择最适当的封装形式。

图5-4中的三个芯片提供的是同样的功能，它们都是AVR ATmega328微控制器。最左边的芯片采用的是直插式封装，它被安装到了一个芯片插座上，无需焊接，因而方便插拔和更换。另外两个芯片展示了采用表面贴装技术的两种封装形式。可以看到，芯片的尺寸明显减小，但这是以增加焊接难度为代价的。

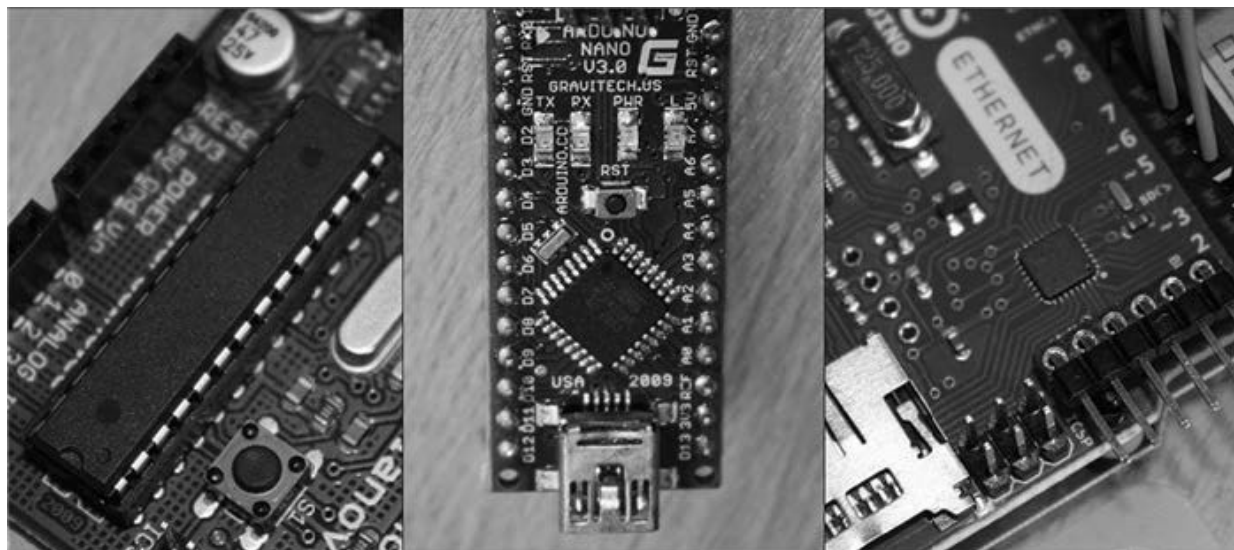


图 5-4 采用直插式封装和表面贴装式封装的ATmega328芯片

借着介绍ATmega328的机会，我们开始对几个特定的嵌入式计算平台进行比较。我们将要介绍的第一个平台曾极大的促进了ATmega328的流行和普及。就在几年前，因为这个平台的流行，导致对直插式封装的ATmega328的需求在短期内超出了供给，使得该芯片在世界范围内发生短缺。

5.3 Arduino

毫无疑问，Arduino是物联网领域乃至更为广阔的物理计算领域的典型代表。

现在的Arduino项目包括了若干种微控制器板。该项目最早于2005年在意大利北部的伊夫雷亚诞生。一个来自于伊夫雷亚交互设计学院（IDII）的小组想为该学院设计专业的学生们找一块单板来构建交互式项目。虽然当时已经有各种各样的单板可供选择，但它们要么价格昂贵，要么不易使用，或者既贵又不好用。

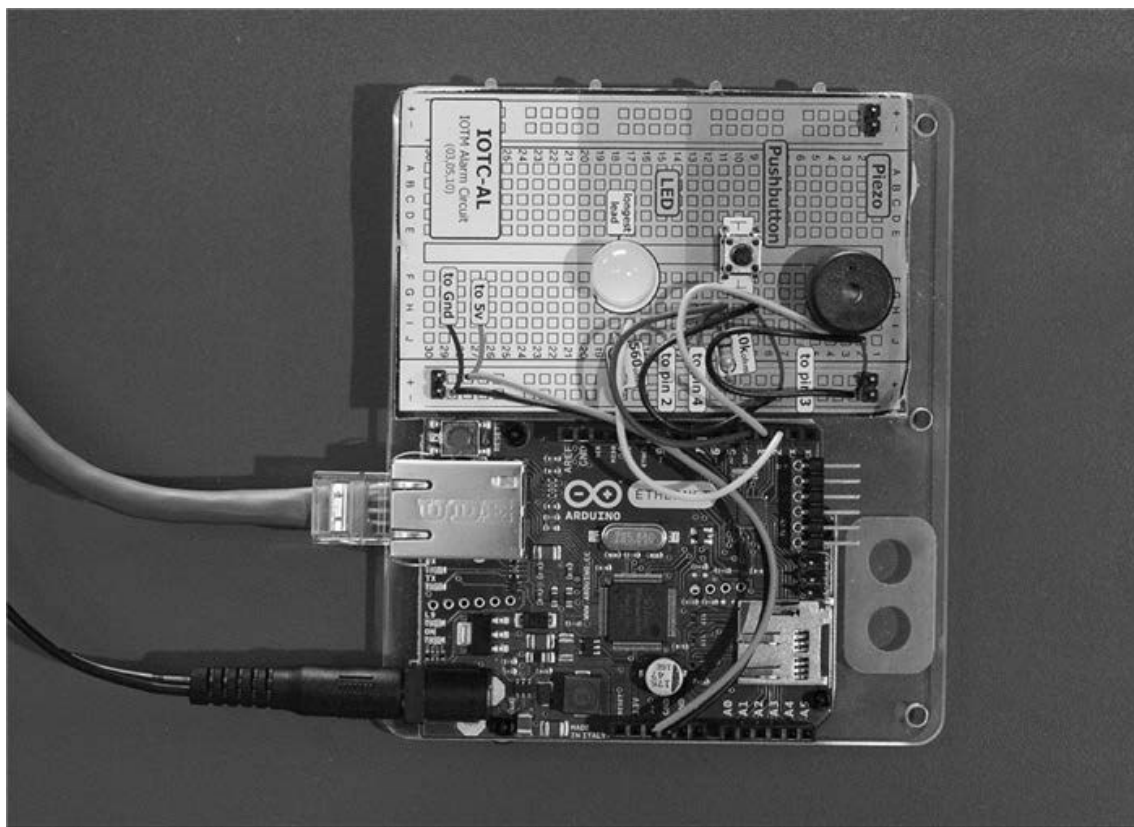


图 5-5 已经连接好外围电路和网络，随时可用的Arduino Ethernet板

于是这个小组就自己设计了一块单板，该单板成本很低，大概在20英镑左右。单板上还设置了一个串行接口，可以很容易地对其编程。这块单板连同Wiring软件环境的一个扩展，在物理计算领域产生了巨大的影响。

Wiring: 为硬件画草图

Wiring是诞生于IDII的另一个项目。2003年夏天，Hernando Barragán创建了一个项目，目的是让用电子装置和硬件做实验的过程变得更加容易。正如该项目的网站 (<http://wiring.org.co/about.html>) 所述:

“我们的想法是：写几行代码，并且连接几个电子元件到所选择的硬件，然后观察当有人接近这个硬件时，灯是如何点亮的；再写几行代码，添加另一个传感器，看看当房间的光照水平下降后，灯的亮度如何变化。

“这个过程被称为用硬件画草图，具体来说就是：选择有趣的想法，对其进行完善，生成原型，通过多次重复这一过程，非常快速地验证很多想法。”

Wiring平台提供了一个在硬件之上的抽象层，使得用户不用操心如何把一个GPIO引脚置为高电平等的具体硬件实现，因而可以把精力集中于他们试图探索或解决的问题。

这种抽象也使得Wiring平台能在各种硬件单板上运行。从Wiring项目诞生以来，已经出现了若干种支持Wiring平台的单板，不过这些单板在一个获得巨大成功的项目面前却黯然失色。这个项目就是基于Wiring平台，采用低端、廉价的AVR处理器的Arduino项目。

由于在项目早期就已经决定对代码和原理图开源，从而使得Arduino板在IDII停止运作之后仍然得以继续存在和繁荣发展，这也意味着人们可以对该平台进行修改和扩展，以满足他们各自的需求。

这样一来，由单板、附加组件和相关工具构成的一个完整的生态系统便蓬勃发展起来。Arduino团队这种聚焦于简单性而不单纯追求代码性能的做法，使得Arduino板几乎成为了每一个初涉物理计算项目的人的理想之选。而Arduino社区的开源风气则鼓励人们共享电路原理图、零部件清单和源代码。现在已经差不多是这样一种情况：不管你在项目方面有什么想法，把你的想法连带Arduino这个关键词在Google上快速搜索一下，你都至少能找到一个有助于实现自己想法的项目。如果更

愿意从书本上学习Arduino，我们推荐这本书：*Arduino For Dummies*。

从Arduino NG、Diecimila、Duemilanove，再到现在的Uno，“标准”的Arduino板已经经历了若干次版本更替。

Uno板的特色是使用了ATmega328微控制器，并且有一个用于连接计算机的USB接口。Uno板有32KB的存储空间和2KB的RAM。不要因为觉得内存太小而感到失望，尽管有内存方面的限制，你还是能用它做很多事的。

Uno板提供了14个GPIO引脚（其中有6个引脚能提供PWM输出）和6个10位精度的ADC引脚。可以直接通过IO引脚或者通过使用附加的芯片和USB连接器与ATmega进行串口通信。

如果你需要更多的内存空间或更多的输入输出引脚，可以考虑选用Arduino Mega 2560。Arduino Mega 2560软件环境没有变化，但硬件方面有很大的增强：使用了功能更强大的ATmega微控制器；提供了256KB的Flash存储空间和8KB的RAM；新增加了三个串口；GPIO引脚的数量多达54个，其中14个引脚能支持PWM；有16个ADC。另外一个选择是使用最新的Arduino Due。这块单板搭载了32位ARM内核的微控制器，是Arduino家族中第一块采用ARM架构的单板。它的硬件规格和Mega板相近，但把RAM增加到了96KB。

5.3.1 在Arduino上做开发

和单纯的参数规格相比，用单板做开发的实际体验可能更为重要，至少在原型制作阶段是这样的。前面已经提到，Arduino在简单性方面做了优化，从单板的配置就能明显地看出来。通过一根USB连线，你不仅能为单板供电，而且还能把代码下载到单板上。如果需要，还能用USB线进行通信，例如，对单板进行调试，把Arduino板所连接的传感器获取的数据存储到计算机中。

当然，尽管Arduino在易用性方面走在了其他平台前面，但本章介绍的大多数的微控制器都在试图做到这一点。只是相对而言，有些微控制器在易用性方面还做得不够好。

集成开发环境

你通常会采用Arduino团队在<http://arduino.cc>上提供的集成开发环境（IDE）做Arduino开发。虽然这是一个基于Processing语言开发环境的功能完善的IDE，但使用起来非常简单。大多数Arduino项目只需要一个代码文件，所以你可以把这个IDE看作是一个简单的文件编辑器。在这个IDE中，用来检查代码（通过编译代码实现）或下载代码到单板的控件是你用的最多的功能。

推送代码

用USB线连接单板相对比较简单。有时，你可能会碰到一些驱动方面的问题（特别是在某些版本的Windows上），或者遇到USB端口的访问权限问题（一些Linux驱动程序包不会把你加入到dialout组），但这类问题一般都能快速地一次性彻底解决。在这之后，你需要选择正确的串口编号和单板类型。你可以通过查看系统日志或者采用试错法找到正确的串口。你需要仔细查看单板上的标签和CPU上的文字，然后从IDE的菜单项中选择正确的单板类型。

如果上述设置都正确，推送代码的过程大体上是比较简单的：首先是对代码进行检查和编译，任何的编译错误都会报告给你。如果代码编译成功，它就会被传送到Arduino板上并被保存到闪存中。此时，Arduino会重新启动，开始运行新的代码。

操作系统

在默认情况下，Arduino本身并不支持在其上运行操作系统。只是用引导程序（bootloader）来简化之前介绍的代码推送过程。单板上电后，就开始运行你编译过的代码，直到单板断电（或代码崩溃）为止。

不过，在Arduino上运行操作系统也是可以做到的。通常可以加载一个轻量级实时操作系统（RTOS），例如FreeRTOS/DuinOS。这些操作系统的主要优势在于其内建的多任务处理支持。但对于很多应用场合而言，使用简单的任务分派库就能获得还算不错的结果。

如果你喜欢稍微复杂一些，可以不使用IDE，而是用适用于AVR芯片的avr-gcc工具集编译代码。在基于ARM的Due板出现之前，avr-

gcc 工具集对于所有的Arduino板都是适用的。

avr-gcc 工具集 (www.nongnu.org/avr-libc/) 汇集了各种程序。它可以用来编译代码，使之能够运行在被大多数Arduino板所采用的AVR芯片上；也可以用来把编译后的可执行代码烧写到AVR芯片上。除了被Arduino的IDE在后台调用外，该工具集也可以被直接使用。

编程语言

通常，Arduino采用的编程语言，是源自于Wiring平台的一个略有修改的C++语言的变种。它包括了若干个库，用来从Arduino板上的I/O引脚读写数据，并能做一些基本的中断处理（以非常底层的方式来实现多任务处理）。这个C++的变种对代码的顺序不做严格要求，例如，允许对某个函数的调用出现在其定义之前。虽然这只是一个细节上的改变，但考虑到代码往往会被全部存放在一个文件中，这种改变使得我们能够以易于阅读和维护的方式安排代码的顺序。

用户代码只需要提供两个函数。

- **setup()**：这个函数在Arduino板上电后只运行一次，可以用来设置I/O引脚的输入输出模式，或者为整个程序都会用到的数据结构做初始化处理。
- **loop()**：这个函数在Arduino板上电期间，会被不间断地反复执行。通常会被用来检查一些输入的状态，在此基础上做一些计算，之后作为响应，可能会改变一些输出的状态。

为了避免在本章中牵涉太多编程语言的细节，我们这里只看一个简单的、在大多数Arduino板上都能运行的例子：让LED灯闪烁。

```
// 在大多数的Arduino板上，13号引脚已被连接到了一个LED灯上。
// 为引脚编号定义一个变量名：
int led = 13;

// 当你按下复位按钮后，setup函数会被执行一次：
void setup() {
    // 把指定的引脚初始化为输出模式：
    pinMode(led, OUTPUT);
}
```

```
// loop函数会被一遍遍地反复执行
void loop() {
    digitalWrite(led, HIGH); // 使LED点亮
    delay(1000); // 等待1秒
    digitalWrite(led, LOW); // 使LED熄灭
    delay(1000); // 等待1秒
}
```

看过这段代码后，你就会发现，**setup()** 函数没做多少事情，只是把13号引脚设置为被控对象（因为该引脚已被连接到了一个LED灯）。

在**loop()** 函数中，LED灯先被点亮，然后又被熄灭。控制LED灯亮灭的电子开关在切换状态之前会有一秒的延时。按照Arduino环境的工作方式，每当一个周期（点亮LED灯，等待1秒，熄灭LED灯，等待1秒）结束因而退出**loop()** 函数时，系统会再次调用**loop()** 函数重复上述过程。

调试

因为C++是编译型语言，如语法错误、没有作变量声明等相当多的错误可以在编译期捕获。因为是在计算机上编译程序，你有充分的机会从编译器获得和问题相关的、详细的、可能对你有帮助的信息。

虽然你需要具备一些调试经验才能够识别某些编译错误，其他的一些错误信息却相当容易理解，例如：

```
Blink.cpp: In function 'void loop()':Blink:21:
error:'digitalWrite' was not declared in this scope
```

在**loop()** 函数所在的21行，我们故意把**digitalWrite** 这个函数名给拼错了。

然而，当代码被推送到**Arduino**上之后，游戏规则发生了变化。因为**Arduino**通常不会连接显示屏，即便发生了错误也很难让你知道。即便代码能被成功编译，还是可能会发生某些错误：无法执行的操作会引起错误，例如除零或试图访问总共只有9个成员的列表中的第10个成员；程序可能会泄露内存，这将导致其最终停止工作。最糟糕的情况是，错误发生时，程序仍然能尽职地工作，但会给出完全错误的结果。

如果**Bubblino**突然不吹泡泡了，我们怎么从以下这些原因中确定问题的真正根源：

- 没人在**Twitter**上提到我们；
- **Twitter**的搜索API已停止工作；
- **Bubblino**不能接入因特网；
- 程序错误导致**Bubblino**宕机；
- **Bubblino**在工作状态，但吹泡泡机的电机发生故障；
- **Bubblino**已关机。

艾德里安喜欢开玩笑地说，他能通过观察**Bubblino**的以太网端口指示灯的闪烁情况，对很多问题进行调试。当**Bubblino**连接DNS，连接**Twitter**的搜索API，或者做其他诸如此类的事情时，这个指示灯都会闪烁。（他还开玩笑地说过，我们不要过分怀疑程序有错，因为电机出故障的主要原因是，哈基姆又把泡沫混合液倒进错误的孔里了。）虽然这个办法可能有助于对上述情况中的两种进行辨别，但对分辨其他几种情况是无能为力的。如果你正在把产品推向大众市场，这个办法就更没什么用了。

WhereDial的第一个商业化版本设置了一组LED灯（共有6个），用作消费者级别的错误排查工具。有错误发生时，这些灯的亮灭模式可以帮助客户解决问题，或者能帮助他们在请求技术支持时对故障进行具体的描述。

捕获运行时产生的编程错误可能是比较棘手的问题，原因是，尽管C++语言有异常处理机制，但`avr-gcc`编译器却不支持此项语法机制（可能是因为异常处理比较消耗内存）。因此，**Arduino**平台不允许你使用常见的`try...catch...`逻辑。

实际上，这意味着你需要在使用数据前先做检查。如果一个数有可能为0，把它用作除数前要先检查一下是否可以这样做。提前检查索引值是否超出边界。为了避免出现内存泄漏，看一下第8章介绍的在嵌入式装置上编写代码的各种小技巧。

通常，完美的代码不是一蹴而就的。在创建代码的过程中，你需要一些手段确定错误的所在位置，从而防止它们再次出现。在没有显示屏的情况下，**Arduino**允许你使用`Serial.write()`，通过USB线输出信息到PC机。虽然你可以用USB线传送各种数据，但调试信息尤其有用。**Arduino**的IDE提供了一个串口监视器，用来显示通过USB线从**Arduino**板传送来的各种数据，包括任何的文本信息，如日志信息、注释等，也包括**Arduino**正在接收和处理的数据的细节信息（目的是对计算过程的正确性进行复查）。

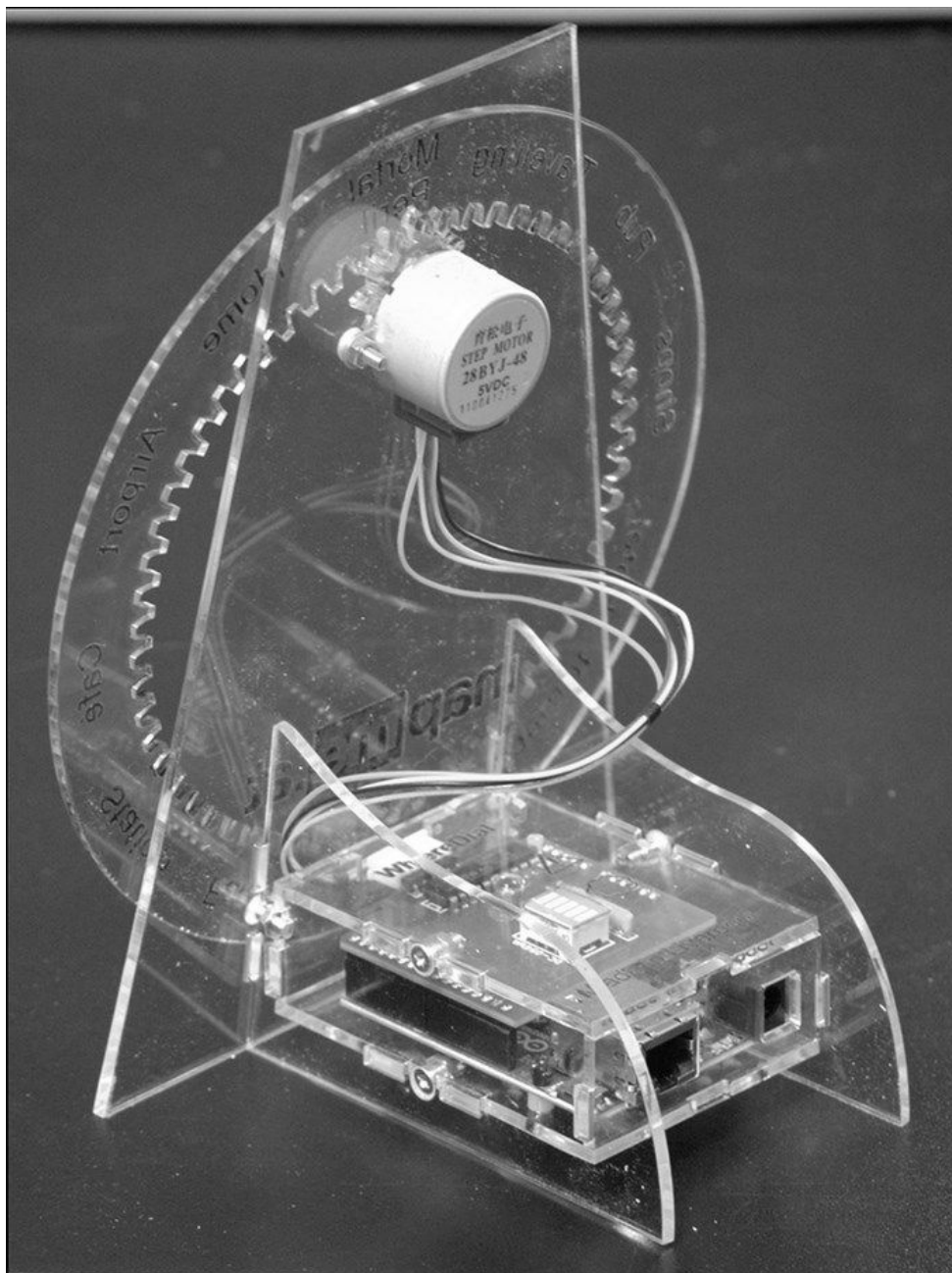


图 5-6 透明材料制成的WhereDial后视图。LED灯组在绿板的中间位置，在红色的“错误”指示灯旁边

5.3.2 硬件相关的一些介绍

Arduino把若干GPIO引脚暴露出来，通常还提供一个连接器（塑料做的条状物，安装在引脚通孔之上，便于实现与导线之间的无焊连接，

特别适用于跳线连接)。这个连接器针对原型制作做了优化，也针对“能够轻松地改变Arduino板的用途”这一需求做了优化。

Arduino板上的每个引脚都有清楚的标记。不同的Arduino板，从比较小的型号，如Nano板，到标准尺寸的Uno板，再到尺寸较大的型号，如Mega板和Due板，它们所提供的引脚各有不同。通常，你会看到电源输出端，如5V或3.3V（一般标记为5V和3V3，也有可能把3.3V标为3V），一个或多个接地端（GND），带有编号的数字量引脚，以及带有前缀A和编号的模拟量引脚。

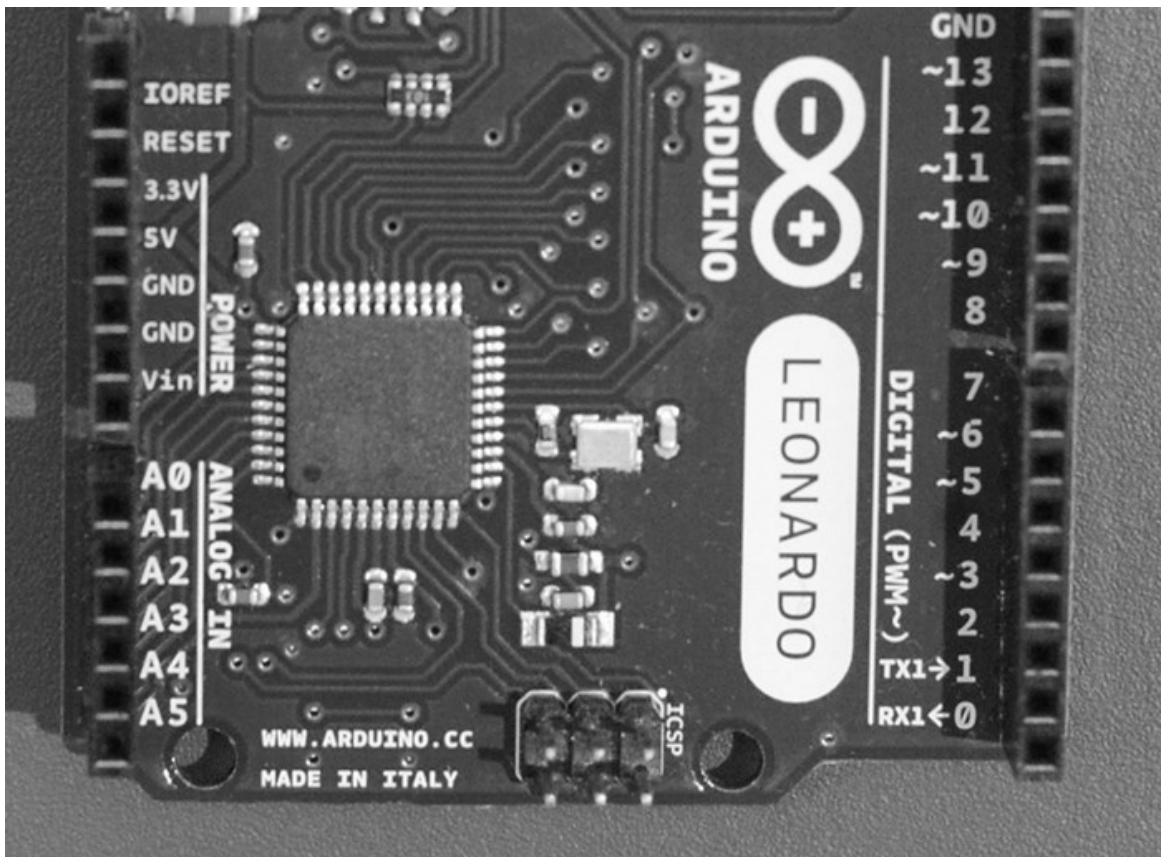


图 5-7 近距离拍摄的Arduino Leonardo板。注意电源和模拟量输入端的标记方式

可以使用USB连接从计算机给Arduino供电。在原型制作阶段，这非常方便，因为总是需要使用串口连接为单板编程。Arduino板也有一个用于连接外部电源的插座，项目发布后你会有更多可能用到它。这两种方式都能为微控制器及其所连接的一些常见的电子器件供电。（对于

比较大的器件，例如电机，你需要为其连接外部电源，在需要的时候，通过使用晶体管为这些器件供电。）

除了标准型号的单板，还有一些专注于特定应用的**Arduino**板。例如，**Arduino Ethernet**板集成了一个以太网芯片，并把**USB**接口换成了以太网接口，使其能够更容易地连接到因特网。显然，这种类型的单板对于物联网项目来说更为适用。

LilyPad板有完全不同的适用领域。它的形状是扁平的（正如它的名字所暗示的一样，它的形状像一朵花，**I/O**引脚的布局有点像是“花瓣”），并在设计上做了考虑，使其能容易地连接导线。对于和可穿戴技术相关的项目而言，这是一块非常有用的单板。

选择一块有特定用途的单板不是扩展**Arduino**能力的唯一办法。大多数**Arduino**板遵从同样的布局，包括了各种**GPIO**、**ADC**和电源引脚。你可以在它们上面叠加一块附加的电路板，而这块电路板可以包含各种元件，以实现附加的功能。

在**Arduino**的世界里，这些附加的单板被称为**盾板**（**shield**）。之所以被称为盾板，也许是因为它们遮盖住了实际的**Arduino**板，看似起保护作用。

一些盾板提供了连网能力，例如以太网、**WiFi**或**Zigbee**等。电机驱动盾板使得连接电机和伺服系统变得简单了。一些盾板像手机一样连接了**LCD**屏幕，一些盾板提供了电容感测能力，一些盾板能够从**SD**卡播放**MP3**文件或**WAV**文件，还有具备其他各种功能的盾板存在。盾板的数量如此之多，以至于出现了一个专门的网站（<http://shieldlist.org/>）来对各种盾板进行比较和记录。

一块叠加了以太网盾板的标准**Arduino**板，从功能角度看，和一块**Arduino Ethernet**板是等效的。不过，后者更省空间（因为所有的组件都集成到一块单板上），但缺少了实用的**USB**接口。（即便使用附带的适配器，也能够通过串口连接**Arduino Ethernet**板，向其推送代码或与之通信。）

5.3.3 开放性

Arduino项目的硬件完全开源，是开源硬件的成功范例。

Arduino商标是该项目中唯一受保护的部分，这样他们就能对任何被称为Arduino的单板进行质量控制。除了代码可以自由下载之外，电路板原理图，甚至EAGLE PCB设计文件也都能在Arduino的官网上轻松找到。

这种共享的文化结出了硕果，各种各样的人们制作出了很多Arduino板的衍生物。有些只是在Uno板的基础上做了细微的修改，但很多单板实现了被Arduino核心团队忽略的需求，引入了新的功能，或者采用了新的外形尺寸。

某些情况下，如聚焦无线传输的Arduino Fio板，开始的时候只是一块由第三方制作的单板（最初的名称是Funnel IO），后来被Arduino团队接纳，成为了通过Arduino认证的单板。

Arduino案例研究：晚安灯

当初还在IDII的时候，Alexandra Deschamps-Sonsino就想制作一种能连接因特网的台灯或床头灯。作为一种简单的供普通消费者使用的装置，这种灯应该能与位于任何地点的另外一个灯配对使用，能远程改变另外一个灯的开关状态，反之亦然。因为灯光已经和我们的日常生活融为一体，看到我们爱的人何时打开或关闭床头灯，能使我们在不经意间了解他们的生活。

这个概念在2005年的时候还是有些超前，但这个项目现在已经发展成了一家公司的产品（Good Night Lamp）。他们的产品由一个大灯和一个或多个小灯构成。大灯有自己的开关，使用方式和普通的灯一样。小灯则没有开关，而是用来显示大灯状态。

在项目的早期，艾德里安就作为首席技术官参与进来。因为艾德里安和团队中的其他成员都熟悉Arduino，所以把Arduino作为原型平台成为一个明显的选择。此外，由于晚安灯不是专用的技术装备，而是面向普通消费者的产品，目标是大众市场，因此产品的设计、成本和易用性也都非常重要。Arduino平台并不复杂，通过权衡最终产品所需的组件，就有可能大幅减少成本和尺寸。

有些消费者不懂技术。对于面向大众市场的联网装置而言，关键的挑战是找到一种便捷的办法，让消费者能把装置接入因特网。即使用户有WiFi可用，在一种没有键盘、也没有屏幕的装置上输入家庭网络的认证信息也是一个挑战。除了研究解决这一挑战的各种选择，并寻找最佳方案之外，“晚安灯”团队也构建了一个通过GSM或3G移动电话网络进行连接的版本。该版本与项目团队的一个愿景相吻合：通过“物理”社交网络，让人们即便在无法用其他方式联网的情况下，也能建立起一种联系。

Arduino案例研究2: Botanicalls

Botanicalls (www.botanicalls.com/) 是技术专家和设计师们的协作成果。它由一套放在花盆中的监测装置构成。如果植物的土壤变得太干，Botanicalls套件就会设法联系它的主人。一篇关于该项目的评述文章幽默地称之为“提升物种间理解的一种尝试。”总之，这是一种在植物的通信协议（叶子的颜色和枯萎）和人类的协议（如电话、电子邮件或Twitter）之间做转换的方式。

该项目最早使用的是一种旧型号的Arduino控制器，目前出售的套件使用的是一块定制的基于ATmega 168微控制器的Arduino兼容单板，仍然采用Arduino IDE编程。为了能和叶子形状的PCB在外形方面匹配，该装置没有使用体积较大的以太网盾板，而是把WizNet以太网芯片集成到了主控板之上。将来的版本很可能会支持WiFi。

Arduino案例研究3: BakerTweet

BakerTweet (www.bakertweet.com/) 实际上是一台适合在面包房中使用的Twitter客户端装置。面包师可能想让顾客在第一时间知道某种食品（如新鲜的面包、热的松饼、覆满糖霜的杯型蛋糕）刚刚出炉，但他想发个Twitter消息通知顾客却是有难度的。因为他所处的环境中存在高温烤箱、面粉粉尘、粘面团和面糊，所有这些都会对电子装置、计算机的键盘和屏幕、平板电脑或手机造

成严重破坏。**Poke**是位于伦敦的一家设计公司，该公司的员工想在第一时间知道，他们当地的面包房在什么时间会新鲜出炉一批他们最喜欢吃的面包和蛋糕，因此他们设计了一台原型装置，以实现这个设想。

BakerTweet使用WiFi通信。面包房通常没有布设以太网线缆，用WiFi正合适。**BakerTweet**用一个针对面包房环境做了防护处理的盒子作为Twitter的客户端。盒子中的电子装置和通用计算机相比，在健壮性方面做得更好。盒子上面有简单的人机交互界面，允许用沾满面粉和面团的手指操作。**BakerTweet**的硬件主要由一块Arduino板、一块以太网盾板和一个WiFi适配器所组成。除了可以简单控制第三方服务（Twitter），该装置也能连接到一个自定义的服务，从而允许面包师自己配置需要发送的消息。

5.4 树莓派

与Arduino不同的是，树莓派设计的初衷更多是用于教育，而不是专门用于物理计算。树莓派基金会的受托人和联合发起人Eben Upton的想法是，构建一台体积小巧且又价格低廉的计算机，就像他小时候曾经用过的计算机一样，可以用来编程和做实验，而不是用来玩游戏。从2006年开始，该基金会聚集了一批人，包括教师、程序员和硬件专家，对这些想法进行了反复研究。

在博通公司（Broadcom）工作的时候，Upton参与了BCM2835 SoC芯片的研发。该芯片配备了一个异常强大的图形处理器（GPU），支持高清视频和快速的图像渲染。该芯片还包含一个主频为700MHz的ARM处理器，这个处理器几乎算是后来加进去的，虽然低功耗、廉价，但还能用。Upton把该芯片描述为“一个嫁接了ARM元素的GPU”（www.gamesindustry.biz/articles/digitalfoundry-inside-raspberry-pi）。

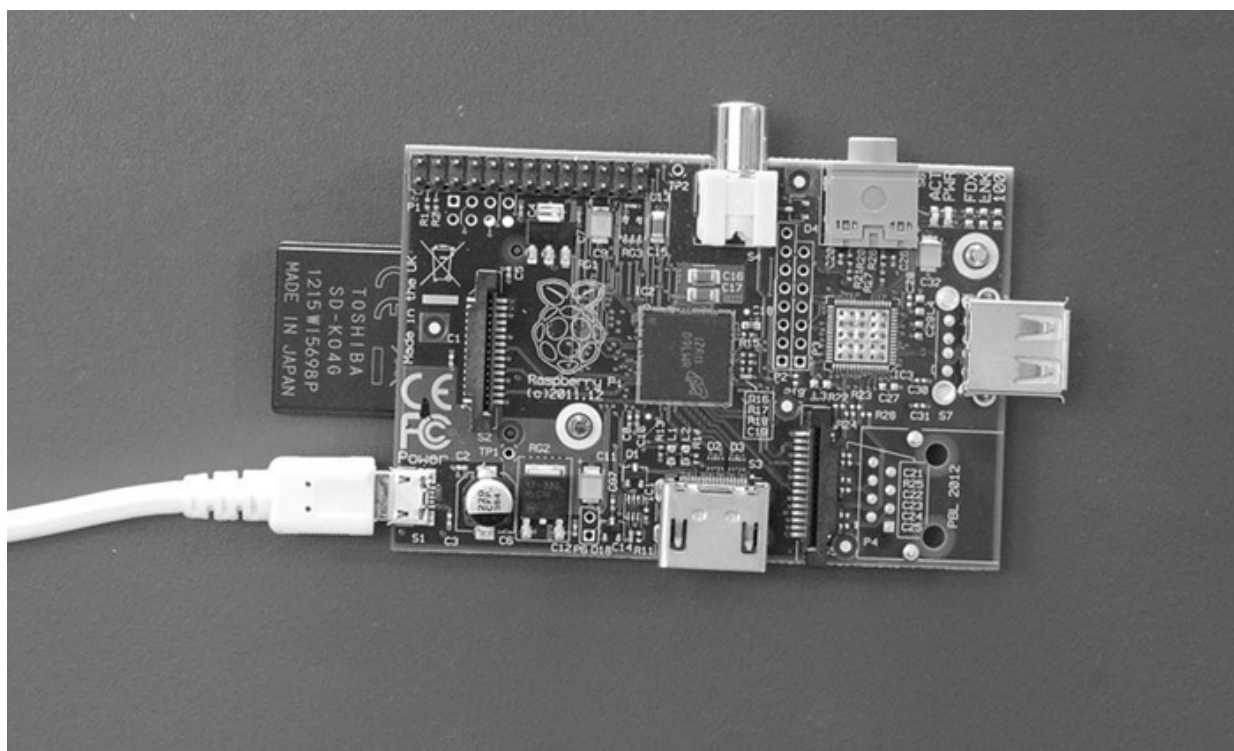


图 5-8 一块树莓派B型板。micro USB连接器只用来给单板供电；USB主机连接器（在底部中间位置和右边中间位置）用来连接USB外设

为改善英国民众对计算机的认知，Acorn公司在20世纪80年代初制造了BBC Micro计算机。树莓派项目就是从这个早期的尝试中获得了灵感。BBC Micro计算机之所以能面世，完全是因为BBC负责制作编程类电视节目的制片人们意识到，当时的英国学校缺少一种能够广泛普及、价格便宜且功能强大的计算机平台。为了使他们的节目成为一个切合实际的选题，他们需要一个这样的平台。树莓派的型号名称A型和B型，正是效仿了BBC Micro对其不同版本的命名方式。在2009年正式创立的树莓派基金会中，有多位成员曾借助BBC Micro积累了开发经验。这其中就包括David Braben，他采用先进的3D线框图形技术，编写了影响深远的星际探索类游戏*Elite*。

尽管树莓派基金会规模不大，但它却能够与大的供应商交涉，压低元器件价格。在很大程度上，这是由于它是一个慈善组织。对于功能比较强的B型板（内建以太网连接），其最终成本大约在25英镑左右。这使得树莓派在价格上与Arduino基本持平，但两者的硬件规格有很大不同。

下表对最新的、功能最强的Arduino Due板和配置较高的树莓派B型板的硬件规格做了对比。

	Arduino Due板	树莓派B型板
CPU速度	84 MHz	700 MHz ARM11
GPU	没有	Broadcom双核VideoCore IV多媒体协处理器
RAM	96KB	512MB
存储容量	512KB	SD卡（4GB +）

	Arduino Due板	树莓派B型板
操作系统	引导程序	各种Linux的发行版本，也有其他操作系统可用
接口类型	54个GPIO引脚 12个PWM输出 4个UART SPI总线 I ² C总线 USB 16U2+原生主机 12个模拟量输入（ADC） 2个模拟量输出（DAC）	8个GPIO引脚 1个PWM输出 1个UART 有2个片选引脚的SPI总线 I ² C总线 2个USB主机接口 以太网 HDMI输出 分量视频和音频输出

因此，树莓派实际上是一台计算机，能运行真正的现代操作系统，连接键盘和鼠标，接入因特网，并向电视或显示器输出高清图像。而Arduino拥有的原始计算能力，以及有限的内存和存储空间，致使它没有足够的资源去运行现代操作系统。重要的是，树莓派B型板有内建的以太网接口（Arduino Ethernet板也有，但Due板没有），还能使用廉价易用的USB WiFi适配器，不需要像Arduino那样使用“盾板”做扩展。

需要注意的是，尽管树莓派的硬件规格，即便和Arduino家族中比较高端的Due板相比，总体上也要强出很多，但不能因此就断定，树莓派要比Arduino好。我们还需要考虑装置的用途。为了说明树莓派在物联网生态系统中的定位，我们需要像在之前介绍Arduino一样，研究一下与树莓派交互的过程，让其像物联网装置那样做一些有用的物理计算工作。下面我们就将介绍这些内容。

不过值得一提的是，市场上还有很多单板与树莓派定位相同，如Chumby Hacker板、BeagleBoard板等，但它们的价格要高出不少。这些单板的确在硬件规格方面更好一点，但考虑到价格的差异，看似没有太多理由考虑树莓派之外的选择。即便如此，项目在平台选择方面，会受到多重因素的影响，例如，现有的硬件条件，对某一特定芯片组能获得更好的工具支持，或者易用性方面的考虑等。为此，我们将在下一节中介绍一块这样的单板——BeagleBone。

5.4.1 外壳和扩展板

树莓派受到了很大的关注，一方面是由于英国主流媒体对它的大肆宣传，另一方面也归因于黑客和创客圈常有的跟风现象。围绕树莓派已经出现了一些生态系统。因为树莓派能被用作通用计算机或媒体中心设备，并且不需要持续不断地用电子元器件制作原型，所以为树莓派找一个方便美观的外壳成了爱好者们第一位的需求。很多创客都在博文中介绍了他们各自的尝试，并且在Thingiverse、Instructables等网站上分享了他们的设计。也有一些和外壳相关的商业项目存在。尽管有树莓派基金会的工作人员通过博文介绍了一个由树莓派标识的设计者Paul Beech早期完成的、设计得不错的外壳

（<http://shop.pimoroni.com/products/pibow>），但并不存在由该基金会授权的官方版本的外壳。这是他们有意为之，目的是促进生态系统繁荣发展，使其尽可能有活力。

除了这些主要和美观度有关的项目外，树莓派的各种扩展板和附件也已面世。因为树莓派从发布到现在也没多长时间，显然其扩展板和附件的数量不能和Arduino相比。不过，很多有趣的配套装置已在开发中，例如，可以方便连接GPIO引脚的Gertboard扩展板

（www.raspberrypi.org/archives/tag/gertboard）。

Arduino的开发者经常会有种似乎什么都已被人做过了的感觉。而树莓派出现的时间不长，局面更鼓舞人心。尽管有很多人在围绕树莓派做各种有趣的事情，但因为这是一个更高级、更强大的平台，所以人们的关注点也比较分散——从外壳设计到操作系统移植，再到媒体中心插件的设计，等等。物理计算只是其中的一个关注点。

5.4.2 用树莓派做开发

Arduino的局限，在一定程度上，也是它最大的特色。与Arduino相比，树莓派平台上能修改的东西要多很多，也更强调能够以多种途径做事情。不过，“最佳实践”当然也在发展形成中。截止到本书写作时，如下的一些建议可供参考。（有必要的話，还可以看看树莓派的官网，相关的IRC频道等途径，了解一下这些“最佳实践”现在的演进情况。）

如果想认真研究一下树莓派，非常建议你找一本由Eben Upton和Gareth Halfacree编著的*Raspberry Pi User Guide*看看。

操作系统

尽管有很多操作系统能在树莓派上运行，但我们推荐使用流行的Linux发行版。

- **Raspbian**：这是由树莓派基金会发布的基于Debian Linux的发行版，是默认的“官方”版本，当然也是树莓派用作一般用途时的一种很好选择。
- **Occidentalis**：这是由Adafruit发布的Raspbian的定制版本。与Raspbian不同的是，该版本假设你以无外设的方式使用树莓派（不连接键盘和显示器），这样在默认情况下你就能远程连接到树莓派。若使用Raspbian，则需要先做一个简单的配置。

对于物联网装置，我们推荐采用Adafruit的发行版。你很可能不打算为装置配备键盘和显示屏，使用这个版本就能在第一时间避免配置的不便（即先要找到需要修改的地方，然后完成各种设置）。该版本让我们感兴趣的主要微调之处是：

- `sshd`（SSH协议的守护进程）是默认允许的，因此你能远程连接到控制台程序；
- 采用零配置网络服务规范（`zeroconf`）对装置进行注册，注册的域名为`raspberrypi.local`，这样你就能用此域名连接该装置，而不再需要知道或者去猜测网络分配给它的IP地址是什么。

在之前介绍Arduino的时候，我们看到，建立开发环境的过程很简单，这可能是其最成功之处。在最好的情况下，只需要把IDE下载下来，再用USB线连接装置和计算机就可以了。（当然，这里略去了和USB驱动有关的古怪问题，以及在制作物联网装置时接入因特网的问题。）而对于树莓派，你需要首先决定选用操作系统的哪个发行版本，并下载该版本。这个发行版需要被解压到一个SD卡（需要你单独购买）。需要注意的是，有些SD卡在树莓派上不太好用，似乎Class 10级别的SD卡最好用。从产品的外包装上不一定能看出SD卡的级别，但在SD卡上能看到一个代表字母C的环形图形，里面有一个代表级别的数字。

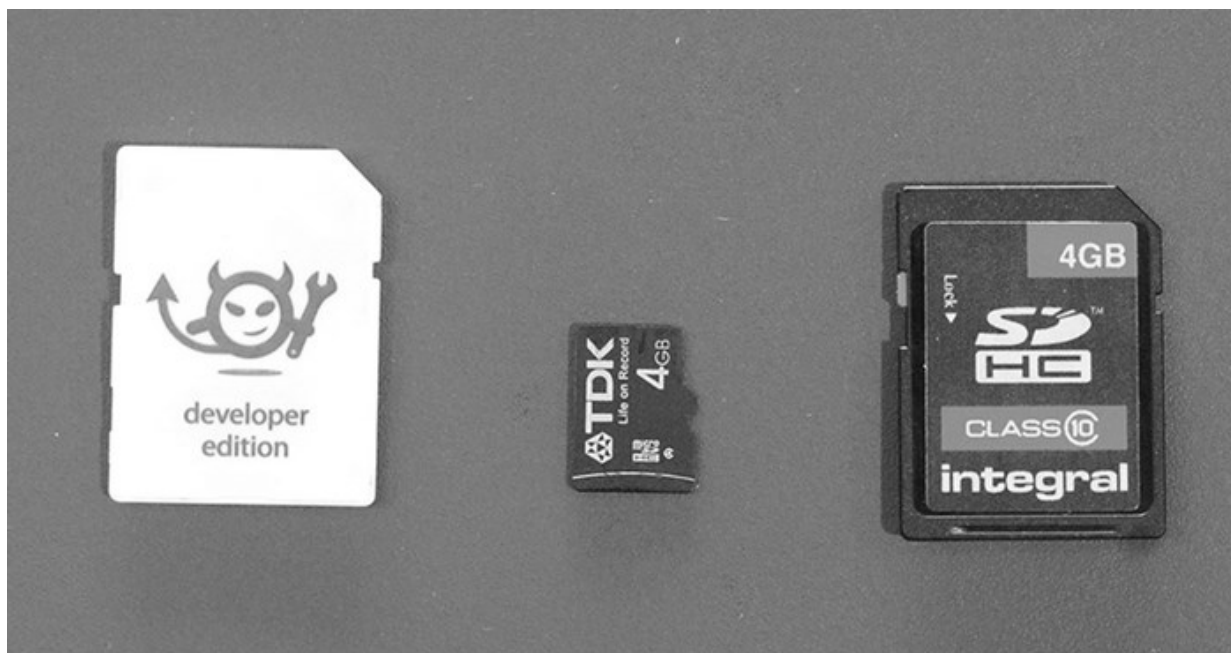


图 5-9 Electric Imp（左）、micro SD卡（中）和SD卡（右）

现在，如果能从USB接口获得足够的电量，你的树莓派可能就启动了。很多笔记本电脑的USB接口存在供电能力不足的情况，此时尽管代表进入运行状态的指示灯是亮的，树莓派却未能成功启动。如果不能确定供电能力是否有问题，使用自带电源的USB集线器看似是最好的选择。

启动树莓派之后，就可以像和计算机通信一样与之通信。既可以连接键盘和显示器进行交互，也可以在使用Adafruit的发行版时，通过前面提到的ssh方式进行通信。在Linux或Mac的命令行界面中键入如下命令，就能登录到树莓派（就像登录到一台远程服务器一样）：

```
$ ssh root@raspberrypi.local
```

在Windows上，你可以使用诸如PuTTY之类的SSH客户端软件（www.chiark.greenend.org.uk/~sgtatham/putty/）。成功连接树莓派后，你就能为其开发软件应用了，其过程如同为Linux计算机开发软件一样容易。至于到底有多容易，则很大程度取决于在Linux上做开发的娴熟程度。

编程语言

你需要对想使用的编程语言和环境做出选择。树莓派基金会在这方面给出了一些指导性的建议。对于教育领域的编程活动，该基金会推荐使用Python语言（实际上树莓派名称中的Pi，其最初代表的就是Python）。

让我们看一下物理计算领域的“Hello World”，即无处不在的“让LED灯闪烁”的例子：

```
import RPi.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BOARD) # 设置GPIO的编号方案，使其与硬件相符

GPIO.setup(8, GPIO.OUT) # 设置GPIO的8号引脚为输出模式

led = False

GPIO.output(8, led) # 初始化LED灯的状态为“熄灭”

while 1:
    GPIO.output(8, led)
    led = not led # 在下一轮循环中切换LED灯的亮灭状态
    sleep(1) # 休眠1秒
```

如你所见，这个例子看上去和Arduino上的C++代码有些类似。唯一真正的差别是模块化的细节：GPIO编号方案，甚至sleep()函数都需要指明。然而，如果能超越这一层复杂性，使用Python这样的表达能力更强的“高级语言”几乎肯定能让下列任务变得简单：

- 处理字符串数据；
- 内存管理（能完全避免自己做内存管理，因而避免出现相关的程序错误）；
- 调用因特网服务并对收到的数据进行解析；

- 接入数据库并做更复杂的处理；
- 提取通用模式或复杂行为。

此外，利用PyPi（<https://pypi.python.org/pypi>）上现成的库，也让复用他人编写、使用和仔细测试过的代码变得简单。

那么，又有哪些不好的地方？与往常一样，你需要意识到，这其中存在一些取舍。这些取舍或者和Linux平台有关，或者和使用的高级语言有关。以后我们提到Python时，相同的考虑因素也适用于大多数其他的高级语言——从Python同时代的Perl和Ruby，到诸如Java和C#这样的编译型虚拟机语言。下面，我们专门把Python和Arduino编程所采用的低级编程语言C++做个对比。

- 和C++相比，Python等大多数高级语言编译后的代码所占内存空间相对较大，运行速度也相对较慢。占用内存不大可能会成为问题，因为树莓派拥有足够多的内存。代码执行速度是不是会成为问题则要看具体情况。Python在执行大多数任务时是足够快的，这其中当然包括任何与接入因特网有关的任务，在网络上进行通信所耗费的时间才是大头。然而，如果你使用的传感器和执行器电路需要非常精确的定时，Python可能会显得太慢。但也不是一定如此。如果bubblino开始吹泡泡的时间晚了1毫秒，或者DoorBot在你扫描RFID卡进行鉴权后，晚了1毫秒才解锁办公室的门，此类延时是可以接受的，甚至是觉察不到的。
- Python会自动对内存进行管理。众所周知，对内存分配的细节进行操控是非常繁琐的，因此自动内存管理通常能减少程序错误的发生，并且能把内存管理得足够好。不过，这个自动化的工作需要在预定时间执行，且要花费时间。根据垃圾收集策略的不同，这可能会导致程序操作的暂停，影响后续事件的定时。此外，因为程序员看不到内存管理的细节，有可能发生这种情况：Python会非常合理地占用较多内存，比你亲自管理内存时使用的内存要多。在最糟糕的情况下，除非进程终止，内存一直都不被释放。这就是所谓的**内存泄漏**。物联网装置通常会在无人值守的情况下长期运行，这样的内存泄漏会逐渐累积，最终会导致装置的内存被耗尽，从而致使程序崩溃。在现实中，出现这样的内存泄漏，更有可能是因为采用了手动内存管理并因此产生了程序错误。

- **Linux**本身在实时应用方面存在一些有争议的问题。由于**Linux**是一个相当大的操作系统，很多进程可以同时运行，因此定时精度取决于在任何给定的时刻，**Python**运行时环境能获得多少**CPU**优先权。虽然这并没有阻止太多嵌入式系统的程序员把系统迁移到**Linux**，但在你的应用中要考虑到这一点。
- **Arduino**只能不间断地反复执行一组指令，直到系统断电或崩溃为止。树莓派一直要运行若干进程。如果其中的一个进程行为异常，或有两个进程发生争用资源（内存、**CPU**、访问某个文件或网络端口）的情况，都会产生一些问题。而这些问题的产生与否，和你的代码是完全无关的。这些问题发生的可能性不大（很多**Linux**计算机和因特网上的其他成员一样，运行着各种商业应用，并且连续运行多年都没什么问题），但却可能导致偶尔出现（或许是间歇性出现）难以识别和调试的问题。

我们当然不是想过分强调上述问题。它们只是一些需要权衡的因素，可能重要，也可能不重要。更确切地说，与树莓派的功能特性和是否使用高级语言的便利相比，它们的重要程度可能较高，也可能较低。

最重要的问题可能还是开发环境的易用性。如果熟悉**Linux**，做树莓派开发相对简单，但还是不如用**Arduino IDE**做开发。例如，**Arduino**一上电就开始运行代码了。而在**Linux**下实现相同的行为，则需要使用若干机制，如在`/etc/init.d`目录下创建一个初始化脚本。

首先，需要创建一个脚本，例

如`/etc/init.d/StartMyPythonCode`。运行该脚本时，如果使用**start**作为参数，则开始运行代码；如果使用**stop**作为参数，则停止运行代码。然后，需要使用**chmod**命令，把脚本标记为系统可以运行的状态：`chmod +x /etc/init.d/StartMyPythonCode`。最后，通过执行**sudo update-rc.d StartMyPythonCode defaults**命令，对该脚本进行登记，使其能在装置启动后即被运行。

如果熟悉**Linux**，可能对这种自动启动服务的机制并不陌生（或许还有更好的替代方法）。如果不熟悉的话，可以搜索一下“**Raspberry Pi start program on boot**”或类似的关键词。不管是哪种情况，尽管完成这些设

置本身并不困难，但如果你之前没有涉足IT领域，还是会觉得要比使用Arduino复杂许多。

调试

尽管Python的编译器也能捕获一些语法错误，并且能尝试使用未经声明的变量，但它也是一个相对宽松的语言（和C++相比），更多的计算是在运行时完成。这意味着有一些编程错误不会导致编译不成功，但会在程序运行时，可能在数天或数月之后，导致程序崩溃。

Arduino的调试能力非常有限，主要包括通过串口输出数据和观察LED灯的闪烁情况。相比之下，运行于Linux之上的Python代码能让你体会到编程语言的优势和使用操作系统的好处。你可以使用Python的集成调试器一步步地执行代码，使用Linux的`strace`命令附着到代码对应的进程，查看日志，观察内存使用情况，等等。只要装置本身还没有死机，你就能够使用ssh登录树莓派，在程序停止工作时（或程序还在运行但功能异常时）做一些调试。

因为树莓派属于通用计算机，不像Arduino那样有严格的内存限制，你能够简单地使用`try...catch...`逻辑捕获Python代码中的错误，并决定怎样处理这些错误。例如，你通常可以利用这个机会对错误的细节进行记录（对调试过程有帮助），并会判断能否对意料之外的问题进行处置，然后继续运行代码。最差的情况下，你可以简单的让脚本停止运行，然后再让它重新运行。

Python和其他高级语言还具备成熟的测试工具，让你能够对程序预期的行为做断言，测试程序的行为是否正确。这种自动化测试能帮你搞清楚自己完成的代码是否正确，并且能在程序做其他修改后再次运行，从而确保对代码某部分进行的修改没有造成其他能够正常工作的部分出现错误。

5.4.3 硬件相关的一些说明

树莓派有8个GPIO引脚，和电源以及其他接口被一同引出，组成一个2×13的双排针区块。与Arduino不同的是，树莓派上的引脚没有被一一标记。这样做是因为树莓派上的组件数量更多，也因为使用GPIO引脚的人估计会比较少，所以树莓派不鼓励把元件直接焊到单板上。这样

做的意图是让你把一个连接器（IDC或类似形式）插到整个双排针区块上，使其连接到一块分接板，再在分接板上用GPIO做实际的事情。

另一种可选的方式是用一端为母接头的跳线连接单独的引脚，跳线的另一端连到面包板。在原理图上能查到各个引脚的功能标注。用一根母对公跳线连接头针和面包板上的插孔是最容易的。如果你只能找到母对母跳线，可以简单的在面包板上插一个排针即可，或者把一根公对公跳线和一根母对母跳线连起来，自己做一根母对公跳线。可以从Adafruit、Sparkfun和Oomlout等面向业余爱好者的供应商，或者从诸如Farnell等规模更大的电子元器件供应商那里买到这些跳线。

排针区块同时提供了5V和3.3V的输出。然而GPIO引脚本身只能承受3.3V的电压。树莓派不具备任何的过压保护机制，如果给这些引脚提供了5V的输入电压，就有损坏单板的风险。可选的两种应对办法是：要么谨慎行事，要么使用具备过压保护能力的分接板。到写作本书时，还没有可供推荐的此类分接板。不过，树莓派官网上介绍的Gertboard看似有望进入推荐名单。

请注意，树莓派没有提供任何模拟量输入（ADC）能力，这意味着在连接传感器时会有一些限制，只能直接连接提供数字量输入（即像按钮一样的on/off输入）的传感器。为了能从光电转换装置、温度传感器、电位计等输入装置获取读数，你需要通过SPI总线连接一个外部的ADC模块。你可以在网上找到具体的操作指导，例如，

<http://learn.adafruit.com/reading-a-analog-in-and-controlling-audio-volume-with-the-raspberry-pi/overview>。

我们之前提到过，给树莓派供电会碰到一些问题。尽管树莓派采用标准的USB线供电，但从笔记本电脑、带电源的USB集线器或USB充电器经由USB线获取的电压会有很大不同。如果无法启动树莓派，就需要检查一下供电能力是否满足需要，并换个电源试试。

5.4.4 开放性

树莓派的用途之一就是用来创建“便于折腾”（hackable）的物件，因此很多组件是高度开放的，例如，诸如Raspbian（基于Debian）之类的Linux定制版本，ARM VideoCore的驱动，等等。这不足为奇。最核心的博通芯片本身是一个专有硬件，博通公司只发布了BCM2835芯片的

不完整数据手册。然而，树莓派核心团队的很多成员是博通公司的雇员，他们一直在积极地创建驱动和诸如此类的程序，而这些程序都是开源的。

这些团队成员已经公布出了一些技术资料，如树莓派单板电路图等PDF文档。然而，对于“它是开源硬件吗”这个问题，目前的答案是“还不是”（www.raspberrypi.org/archives/1090#comment-20585）。

树莓派案例研究：利物浦DoES创客空间的DoorBot

当包括我们在一帮朋友和合作者们打算建立一个办公场地和创客空间时，我们就很快发现，为了让这个空间保持开放状态，需要有一个管理钥匙的人早到晚走，这样才能吸引成员前来。由于这里有足够多分文不取的组织者，事情一直进展得很顺利。但在DoES运作了一年半后，John和Ben对DoorBot做了扩展（我们在第4章提到过），在入口处使用了磁控锁。

现在的DoorBot运行在树莓派平台上。三套DoorBot装置各自控制一个门（通往主空间、办公室或举办活动的空间）。它们从RFID读卡器获取输入，然后把输入信息与数据库中已知的身份信息进行比对。如果身份核实成功，DoorBot会播放一段该用户专属的“入场音乐”，并向磁控锁发送一条命令，使门打开3秒，还会对该用户进入空间的行为做日志记录。这个方案需要安装带控制线的新锁，还需要在墙板上钻孔，把RFID读卡器装到外面（我们买的读卡器，虽然价钱合理，但却不能隔着玻璃读卡）。DoorBot还配备了一个显示屏，用来显示活动日程和一些来自于网络摄像头的内容。音频和可视化方面的需求是选用树莓派实现该系统的主要原因。不过，通用的Linux系统在一般情况下也是有用的。在门禁失效的情况下，可通过ssh远程连接树莓派把门打开。

值得指出的是，树莓派用到的这块博通芯片现在很难获得。相比之下，Arduino用的Atmel公司的芯片和BeagleBone用的TI公司的芯片则到处都有。这可能会使原型到产品的转化过程变得更加困难。

5.5 BeagleBone Black

BeagleBone Black是出自BeagleBoard团队的一个最新装置。该团队主要是由TI公司的雇员组成。尽管该产品本身不是TI公司的单板产品，但在他们的雇主的许可下，开发团队使用了很多TI的元器件。这种关系类似于树莓派基金会和博通公司的关系。同样，BeagleBoard团队也想创建“功能强大、开放的嵌入式装置”，以为开源社区做贡献，为电子学方面的教育提供便利。不过，他们的设计重点没有放在创建通用的、适用于教育领域的计算装置上。各种单板在设计的时候就有明确的预期，即它们将被用于物理计算和电子学实验。

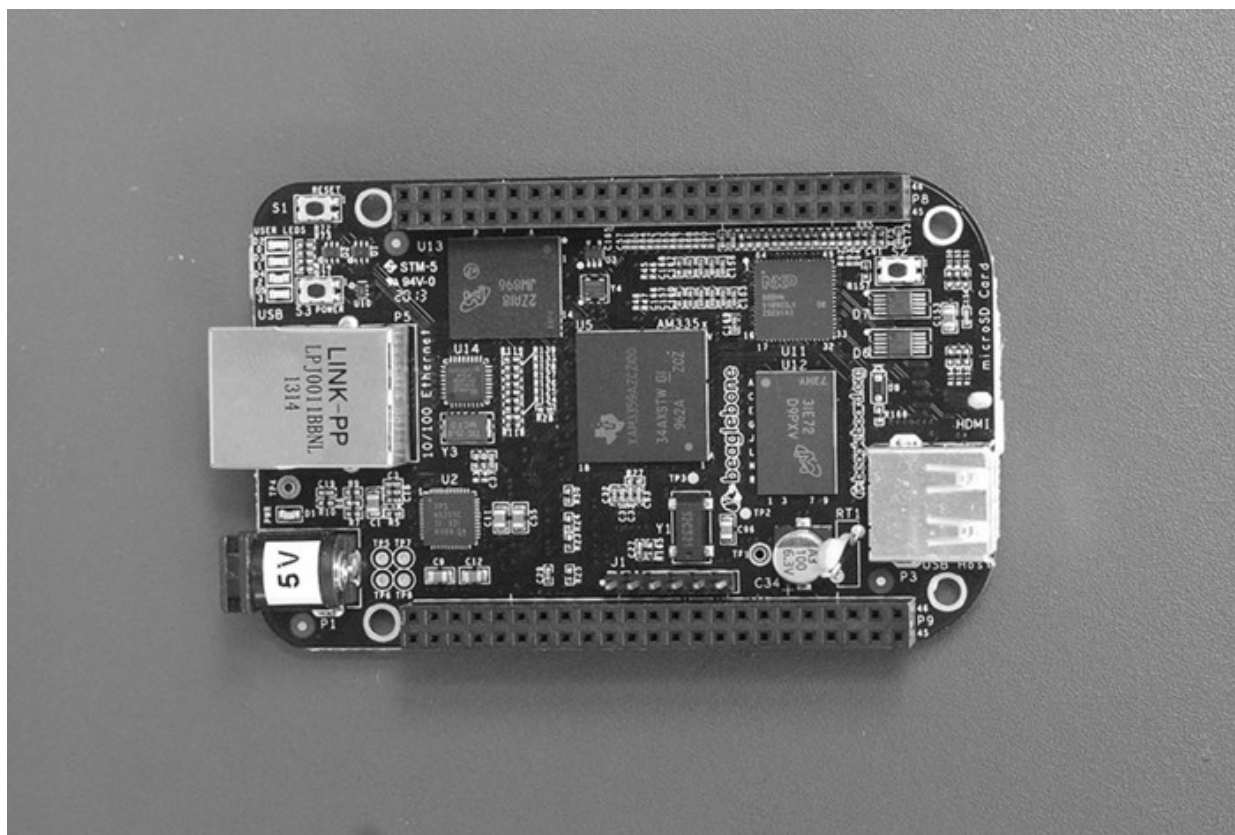


图 5-10 BeagleBone家族最新的单板：BeagleBone Black

在BeagleBoard团队设计的单板中，BeagleBone Black是最小巧便宜的。它的外形尺寸和树莓派差不多。尽管这两种单板的规格大体相似，但还是存在一些有趣的差异之处。

最早的BeagleBoard板没有内建的视频或音频输出，但却拥有数量众多的GPIO引脚，并且以双排母座的形式引出。它还具备至关重要的用于模拟量输入的ADC引脚，而这正是树莓派所缺失的。由此可见，开发团队专注于构建能与电子器件协同工作的装置，而非设计通用的计算装置。

BeagleBone板的发布时间要早于树莓派，而它的售价也反映了这一点。如果你认为它是一块功能强大的嵌入式开发板，比任何一块Arduino板都要强，那么63英镑售价看上去蛮合理的。然而，当你拿它和售价只有25英镑的树莓派做比较，它的售价就显得不太合理了。

从最新版本的BeagleBone平台，BeagleBone Black，可以看出树莓派带来的影响。尽管它还是缺少模拟视频和音频连接器，但却增加了micro-HDMI连接器，用来提供数字音视频输出，其售价也降到了与树莓派很接近的31英镑，并且还保留了原来BeagleBone所具有的、比树莓派要好得多的与电子器件的接口能力。

让我们对树莓派B型板和新的BeagleBone Black板的规格做个比较。

	BeagleBone Black板	树莓派B型板
CPU 速度	1GHz ARM Cortex-A8	700 MHz ARM11
GPU	SGX530 3D	Broadcom双核VideoCore IV多媒体协处理器
RAM	512MB	512MB
存储 容量	2GB eMMC + microSD卡	SD卡（4GB+）
操作 系统	各种Linux的发行版，Android，也有其他操作系统可用	各种Linux的发行版，也有其他操作系统可用

	BeagleBone Black板	树莓派B型板
接口类型	65个GPIO引脚，其中有8个支持PWM 4个UART SPI总线 I ² C总线 USB 从机+主机 以太网 Micro-HDMI输出 7个模拟量输入（ADC） CAN总线	8个GPIO引脚，其中1个支持PWM 1个UART 有2个片选引脚的SPI总线 I ² C总线 2个USB主机接口 以太网 HDMI输出 分量视频和音频输出

和树莓派一样的是，BeagleBone是一台主要运行Linux的计算装置，但BeagleBone也能运行其他各种经过移植的操作系统。与树莓派不同的是，通过显示装置和键盘与本地用户交换信息并不是它的默认选项。

这两种单板都配备了以太网接口（采用树莓派B型板），并且在需要的时候能利用廉价的USB WiFi适配器连网，这对物联网装置而言是非常重要的。

如果BeagleBone的实际优点只是CPU速度稍快一点，（在我们看来）外观设计更加美观一些，就没有充分的理由专门辟出一节来介绍这个平台，更不要说实际购买了（考虑到它的售价）。

让我们现在就对BeagleBone有别于其他平台的特性做个介绍，包括硬件、软件和可用性相关的特性。

5.5.1 外壳和扩展板

尽管BeagleBone没有像树莓派那样进行高调宣传，增加媒体曝光度，但我们还是可以获得相当数量的BeagleBone外壳。这些外壳或者像Adafruit BoneBox那样作为产品出售，或者以设计指南和设计方案的形式被免费提供。不管怎样，考虑到BeagleBone单板主要是为开放硬件设计的，对于你正在从事的项目，为了能和项目作品的外形尺寸相匹配，你也很可能需要一个定制的外壳。

BeagleBone的扩展板一般被称为cape，而不是盾板（Arduino扩展板的名称）。cape这个称谓源自于20世纪60年代的一部美国动画片，其中的主角是一条名为Underdog的穿超人斗篷的猎犬。为LCD屏幕、电机、各种联网和传感器应用增加控制器模块，都有相应的扩展板可用。

5.5.2 在BeagleBone上做开发

我们已经看到，在Arduino上做开发有一套标准的方式，而树莓派则提供了最大的自由度，你需要自己选择操作系统和编程语言。在Arduino上做开发时，你的确可以完全忽略IDE的存在，不使用Arduino的库，而是直接使用avr-gcc，编写代码并直接针对AVR芯片构建一条工具链。然而，几乎所有的Arduino用户至少在开始阶段会使用官方的开发工具，其中只有少数人会尝试其他途径。相比之下，对于在树莓派上做开发，尽管存在各种推荐方案和最佳实践，但你必须明确选择用哪套工具，然后再把它们安装到SD卡上。这种灵活性的存在有一定道理，因为树莓派的目标用户群体非常广泛（涉及教育、通用计算、编程、电子和家庭多媒体等多个领域）。

尽管BeagleBone也能胜任通用计算平台的角色，但它的目标市场比树莓派要窄得多。正因为这样，BeagleBoard团队选择了一条中间路线：为每块单板预装Ångström Linux发行版。尽管其他几种操作系统也已经被移植到了BeagleBone平台，但有预装的默认系统也意味着开始使用和评估单板的过程会变得更简单。如果你喜欢捣鼓，有更喜欢的嵌入式操作系统可用，或者让Ångström达到了某个不可逾越的极限，当然可以去安装另一个操作系统。不过，在本章中，我们只介绍默认安装的系统。

BeagleBone默认支持零配置联网，其在网络内公布的域名通常为beaglebone.local。你可以用若干种方式连接到BeagleBone。你可以连接到一个非常有用的、极好的、交互式的系统参考手册和硬件文档页面，其地址通常是[http://beaglebone.local/ README.htm](http://beaglebone.local/README.htm)。你也可以在<http://beagleboard.org/static/beaglebone/latest/README.htm>找到这个手册，不过在这种情况下就缺少有用的互动效果了。

这个手册的页面会做动态更新，从而给出BeagleBone板的最新信息。你可以在其中的一些页面上查看和修改各种设置，如GPIO引脚的输

出。页面上还有到Cloud9 IDE的链接。这个IDE位于BeagleBone板之上。如果要为该装置编写代码，使用这个IDE是最简单、推荐采用的方法。

Cloud9是一个在线编程环境，也是一个相当有趣的理念。它的一个在线托管的版本位于<https://c9.io>，你可以“在云端”编辑代码（即看作一项因特网服务）。不过，该服务的主要组件是一项开源应用，能够在本地任意指定的一台计算机上运行（<https://github.com/ajaxorg/cloud9/>）。因为BeagleBone本质上是一个通用计算装置，使用可在本地独立运行的版本更合乎情理。这意味着，你完全不需要为了在BeagleBone上做开发而去下载任何软件，只要连接到位于<http://beaglebone.local:3000>的IDE，就可以立即开始工作。

尽管在线版本的Cloud9环境也支持Ruby和Python，但免费组件版本只支持Node.js。Node.js是一个建立在JavaScript之上的框架，在其官网上有如下介绍。

Node.js是基于Chrome JavaScript运行时建立的一个平台，用来轻松地构建快速、易于扩展的网络应用。Node.js采用了事件驱动和非阻塞I/O模型，使得它具有轻量级和高效的特点，非常适合于在分布式环境中的装置上运行数据密集型实时应用。

——<http://nodejs.org/>

我们之前已经介绍过怎样在“实时”平台和运行在树莓派上的Python之间进行权衡。很多相同的考虑因素也适用于Node.js平台，因此这里只做一个简单的归纳。

- 尽管已对Chrome JavaScript运行时做了大量优化，但它还是一个虚拟机，运行的是一种非常高级、占用内存较多的动态语言。
- 自动化的内存管理存在一定的开销，可能会影响定时精度。
- 和真正的实时操作系统相比，Linux本身不太适合实现高精度的定时。

在谈论实时编程时，多任务模型是需要权衡的另一个重要因素。如果一个以上的任务需要在“同一时间”（或看上去是在同一时间）执行，例如，让电机运行，检测按钮是否被按下，让多个灯以不同的模式闪烁，运行时环境需要一个选择策略，用于确定何时在“同时执行”的各个任务间做切换。**Node.js**使用的是一个协作模型，在一个中央进程的协调下，各个操作以适当的方式被执行。这个中央进程基于它的调度系统，选择下一个要执行的操作。但因为一个时刻实际上只能执行一个操作，该进程的协调器需要等上一个操作执行完毕才能开始执行下一个操作。这意味着没有办法能保证把某个任务的开始时间精确到毫秒级。

Node.js的官网提到了“实时”代码，这实际是想暗示，它的开发者们认为他们的技术能很好地实现多任务处理和事件调度。尽管他们对“实时”的定义可能和某些从事高精度实时应用开发的嵌入式程序员想的不一樣，但对于大多数任务已经足够了。更何况，高级语言的优点和方便的开发环境通常是被最优先考虑的。

之前提到过，**Arduino**的一个优点是只有单一的标准开发工具链，而树莓派则利弊兼有，面临选择过于丰富的窘境。**BeagleBone**结合了这两种极端情况的优点，在具备树莓派的全部灵活性的同时，只安装了一个单一的标准工具链。

本节的后续部分将主要介绍标准的工具链。不过，请放心，如果该工具链不能满足你的需求，你还有很多可以替换的选择。

操作系统

BeagleBone在出厂时已经预装了**Ångström Linux**发行版。该操作系统是专门针对嵌入式装置开发的，只要有4MB的闪存就能运行。

与树莓派上的**Occidentalis**十分相似，**Ångström**被配置为以“无外设”方式运行，不需要键盘和显示器。通过**USB**串行连接，或者使用零配置联网方式并连接到**beaglebone.local**，都可以方便地使用命令行接口。

预装操作系统的好处之一是，你开箱之后就可以马上开始使用单板。当然，你会看到操作提示，作为首先要完成的步骤之一，如果当前的版本不是最新的，建议先更新操作系统。当你对单板做完简略的评

估，在准备继续用它构建项目的原型之前，把操作系统升级到最新版本当然是一个好的做法。

编程语言

和树莓派一样，你可以使用终端程序连接到BeagleBone，之后开发软件应用的步骤，和在任何其他的Linux计算机上做开发是一样的。在上一节中，我们已经指出，如果你已经熟悉Linux编程，在BeagleBone上做开发就是一项简单的任务，否则就有可能觉得有点崩溃。在本节中，我们将对之前提到过的Cloud9 IDE做更详细一些的介绍。

我们依旧还是从老套但值得信赖的“让LED灯闪烁”这个例子开始介绍。在README.html页面中，给出实现这个例子的代码：

```
require('bonescript');

setup = function () {
    pinMode(bone.USR3, OUTPUT); // 允许对LED灯USR3进行控制
}

loop = function () {
    digitalWrite(bone.USR3, HIGH); // 点亮LED灯USR3
    delay(100); // 等待100ms
    digitalWrite(bone.USR3, LOW); // 熄灭LED灯USR3
    delay(100); // 等待100ms
}
```

与之前的例子相比，代码的编写思路实际上没什么变化。虽然语法细节有所不同，但还是需要实现**setup** 和**loop** 两个函数，需要把引脚设置为输出模式，然后以一定的时间间隔向该引脚交替地发送HIGH和LOW两个值。

注意，USR3实际上已经连接到了单板附带的一个LED状态指示灯上。如果你想使用GPIO引脚连接一个普通的LED灯，只需要把USR3替换为一个普通的GPIO引脚，如P8_3。

和Python一样，JavaScript是一种高级语言。当你处理完与电子器件之间的物理输入输出，开始做文本处理、与因特网服务交换信息等工作时，用JavaScript做这些事情要比在Arduino上用相对低级的C++做容易很多。

此外，Node.js是一个资源丰富的开发环境，有很多的库，可以集成到应用中来。目前，方便易用的npm（Node Packaged Modules）包管理器还没有集成到IDE中，但未来的版本会将其加进来。与此同时，在线帮助和论坛应该能够帮助你克服任何可能碰到的问题。

与Arduino不同的是，系统不会在重启之后自动运行代码。不过，一旦你完成应用开发，就可以把它复制到autorun子文件夹，这样系统在重启之后就会自动运行该应用。

调试

和Python一样，JavaScript是一种语法比较宽松的语言（和C++相比）。同样因为它是一种动态语言，编译器不会捕获某些类型的编程错误。幸运的是，IDE能报告运行时错误，这很有用。因此，当你在IDE中运行程序时，你能轻松地获得基本够用的诊断信息。

当你开始让Node.js代码以服务的形式自动运行时，你可能需要使用日志，对那些数天或数月之后才能显现的错误进行捕获。

和树莓派一样，Linux操作系统上所有的工具你都可以使用，用来帮助你调试程序。这样做当然很有必要，因为Linux + Node.js是一个比Arduino更复杂的平台。正因如此，其灵活性和强大功能也颇受青睐。

此外，JavaScript有错误处理机制。尽管该机制在被扩展之后，可以说是被Node.js的回调机制复杂化了，但还是有助于优雅地对异常情况进行处理和恢复。

Node.js也具备诸如node-unit之类的自动化测试工具。用高级语言编程并不意味着代码会变得简单。能比较轻松地编写任意数量的代码通常意味着你要编写更多更复杂的代码。因为Node.js是通过回调机制实现的协作多任务处理，程序的执行路径可能会特别复杂。测试过程能让你确保代码中所有预期的路径都能以正确的次序被执行。

5.5.3 硬件相关的一些说明

因为主要面向物理计算领域，BeagleBone单板上可用的GPIO引脚数量要比树莓派多得多，其中一些单板配备ADC（使其能处理模拟量输入，而树莓派则不具备这项能力），另一些单板提供了PWM能力（使其能近似的提供模拟量输出）。尽管各个引脚都沿着单板的长边，每个边两排，整齐地排列在单板的两侧，但BeagleBone还是没有足够的空间，像Arduino那样，对每个引脚都做标记。不过，每一组引脚都有标记，其中的每个引脚的功能在文档中都有介绍。和树莓派相同的是，根据使用该引脚的子系统的不同，每个引脚可以对应不同的名称。如果习惯了Arduino上简单而受限的命名（标记）规定，可能会觉得这样做过于复杂了。

5.5.4 开放性

BeagleBoard团队从一开始就有一个明确的目标，即为了便于开展嵌入式硬件方面的实验，创建一个开放的平台。因此，和树莓派基金会相比，该项目迄今为止以开源的形式发布了更多的原理图。确切地说，所有来自于BeagleBoard.org的硬件都是开源的，因此你可以下载到各种设计资料，包括原理图、材料清单和PCB布线

图”（<http://beagleboard.org/>）。在BeagleBone上预装的Ångström Linux发行版也是完全开源的。beagleboard.org上大多数的非技术性内容也是以创作共享许可协议（Creative Commons Licence）的形式发布。尽管树莓派团队看上去也很支持开源，但BeagleBoard团队似乎更重视能把各种资源明确地以开源许可的形式发布出来。这种做法可能会对你的项目有重大影响。除了哲学理念方面的原因，在你的项目处于从原型到产品的过渡阶段时，原理图和代码的开放可能是至关重要的。

BeagleBone所在的网页上，也有到专有操作系统和扩展板的链接。如你所料，我们认为这是好事。Beagleboard团队正在努力促成一个围绕其产品的良好生态系统，不管相关的产品是免费的还是盈利性的。

BeagleBone案例研究：忍者块

随着我们正在介绍的这些廉价微控制器的兴起，实现诸如家庭自动化之类的物联网应用的成本，已经降到了有兴趣的爱好者们所能接受的价格范围。同时，越来越简单的开发环境使得那些没有

或只有很少编程经验的人得以上手做开发。尽管如此，使用Arduino、树莓派和BeagleBone之类的单板开发有用的原型时，还是需要你自己动手，直接应对各种棘手的软硬件相关的细节。忍者块（Ninja Blocks，<http://ninjablocks.com/>）正是针对这个问题的一个解决方案。正如IFTTT（If This Then That）为Web应用提供了简单的基于规则的编程方式——“如果我在Facebook上标记了一张图片，那么就把这张图片上传到Dropbox”——Ninja Rules应用把这一理念扩展到了物联网装置：“如果我的洗衣机已经把衣服洗好了，那么就给我发个电子邮件。”

忍者块的核心部件是一块运行Linux的BeagleBone单板和一块Arduino板（大概是为了利用它的接口，尽管BeagleBone有大体相当的连接选项）。忍者块能够连接到提供输入输出功能的其他块。输入能力由传感器块提供，例如，用于安全应用的动作和门接触传感器，常规的家庭监控用到的按钮和温度传感器。输出能力由执行器提供，目前能实现对电源的远程控制。尽管用Rules应用构建用例很方便，但你也能使用REST API给忍者块编程，或者，直接在BeagleBone上使用高级语言（如Node.js）或在Arduino上使用低级的C++语言编程。

5.6 Electric Imp

虽然我们将在这里以较大的篇幅介绍Electric Imp，但它还不太成熟，和我们讨论过的其他单板相比，在某些方面存在更多问题。在写作本书的时候，我们也不能断定这个平台是否能发展成为一个可供选择的平台。但它值得我们对其进行比较细致的讨论，因为对于着手研究消费电子和物理计算的开发者来说，该平台的实现方式可能是在解决问题方式上的重大变革。

Hugo Fiennes曾经是苹果公司iPhone团队的技术经理，他想尝试在LED灯的状态和谷歌公司的股价之间建立联系。他对各种实现家庭自动化的可选方案进行了评估，如ZigBee，但却发现它们基本上都是由单一供应商提供的解决方案，通常使用的是它们自己的无线电标准，而不是建立在开放平台的基础之上

(<http://www.edn.com/electronicsnews/4373185/Former-Apple-Google-Facebook-engineerslaunch-IoT-startup-item-2>)。Electric Imp使用了现有的若干标准（如WiFi），其外形尺寸与SD卡相同。和本章中介绍过的所有其他装置相比，这个平台相对是最封闭的。Fiennes在这个项目上的合作者是原Gmail设计师Kevin Fox和固件工程师Peter Hartley。如你所见，这家初创企业好像继承了来自于iPhone和Gmail这两个技术精湛的封闭系统的许多“基因”（包括好坏两方面）。

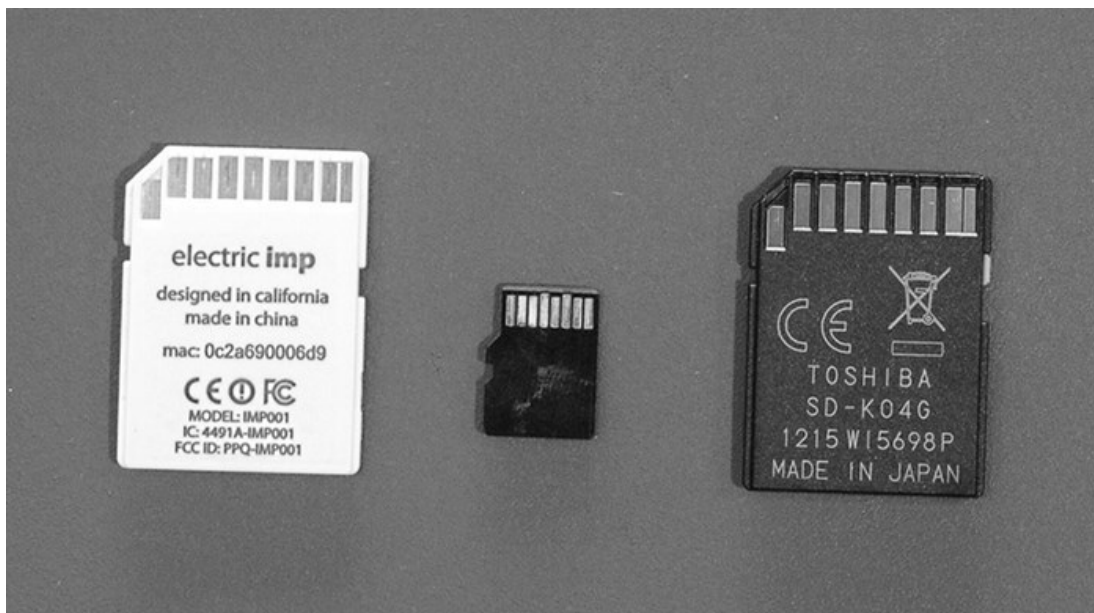


图 5-11 Electric Imp、micro SD卡和SD卡的背面

Electric Imp所有的智能和WiFi联网能力都是由一个SD卡形状的微控制器实现的。需要注意的是，Imp实际上不是SD卡，只是外形相像而已。使用与SD卡相同的外型尺寸意味着Imp的生产成本会比较低廉，因为现有的外壳和工具都可以重复利用。此外，现有的用于连接impee（在Electric Imp平台上，把Imp插入后能与之建立连接的其余电路被称为impee）的组件连接器也都能重复使用。正如你将要在“开放性”小节中了解到的，这最后一个因素很重要。

尽管SD卡感觉上去经久耐用，但从外表来看，实际就是一个小塑料片。SD卡只提供了一种连接方式，即把它插入一个装置。正如SD卡采用插入的方式连接音乐播放器、计算机或打印机一样，你可以把Imp插入impee。impee作为宿主单元，为Imp供电，提供到传感器和执行器的GPIO连接。impee还提供了一个身份识别芯片，以便让Imp知道它插入了哪一台装置。

Imp的价格约为20英镑，而impee比Imp便宜一半还多。对于原型制作者来说，采用标准SD卡的外形尺寸被证明是一个明智的选择。同时开展几个原型制作项目时，只需要一个Imp。它可以在所有项目中重复使用。你很快将会看到，对Imp进行重新配置，使其在不同的impee中运行，这一过程是自动完成的。这是一个很不错的功能。

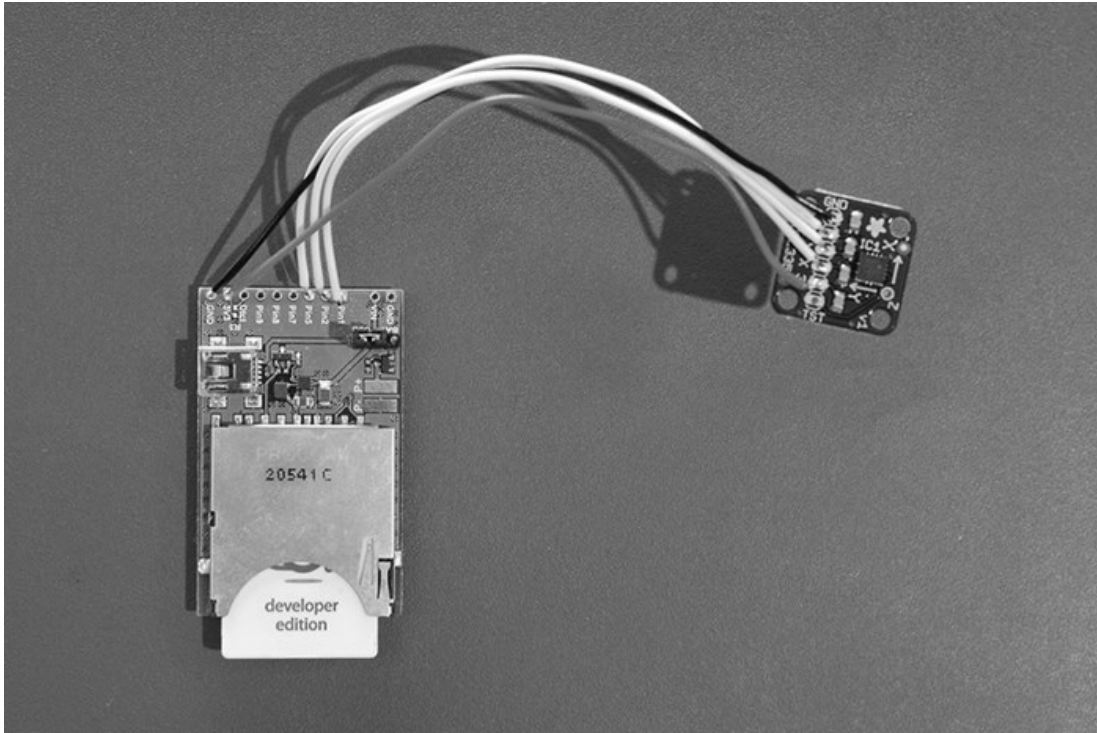


图 5-12 一块Electric Imp被插入了April impee开发板中，并与加速计（较小的那块电路板）相连

在Electric Imp上做开发

我们看到，Electric Imp从impee获得供电。诸如标准的April impee这样的单板有专门的引脚用来焊接电池，同时也提供了一个USB连接以方便开发。但与我们介绍过的其他单板不同，这个USB连接只能用来供电，不能与其他装置通信。所以，你可能会感到奇怪，怎样才能对Electric Imp编程呢？

实际上，对Imp编程和控制是通过因特网实现的，因此你要做的是利用它内建的无线连接去连接WiFi网络。当然，你还需要告诉它具体的无线接入点，很可能还要提供密码。在你找到一种与Imp交换信息的方法之前，这显然是个棘手的问题。Electric Imp团队采用一种被称为BlinkUp的巧妙方案解决了这个问题。Imp有一个内建的光敏元件（仍然包含在小巧的SD卡外形之内），每当Imp无法接入WiFi网络时，该光敏元件就会被激活。在iPhone和近来发布的安卓智能手机上，可以安装一个Electric Imp应用程序。该程序能通过改变屏幕的颜色，交替显示黑白两种颜色，实现对WiFi连接参数的编码。把手机屏幕面向

Imp放置，实际上就可以向这个微控制器发送信号，实现与Imp的通信了。如果你没有适当型号的手机可用，那就不太好办了（当前还没有其他替代的方法可用，例如，通过计算机屏幕的亮度变化对数据进行编码），但我们认为你在需要的时候能够借到一部这样的手机。当然，向Imp发送WiFi连接参数的过程可能会失败，Imp会通过LED灯的某种闪烁模式对此做出响应，帮助你诊断问题。

1. 编写代码

你在Electric Imp官网上能看到一个注册链接。注册并登录账号（<https://plan.electricimp.com/>）后，你会看到一个所有装置节点的列表，开始时列表是空的。你可以在这个IDE中编辑代码。

该平台采用的编程语言是Squirrel。这是一种语法类似于C语言的动态语言。目前，Squirrel的文档还不完备，语言本身还需优化。Squirrel和Lua等高级嵌入式脚本语言有一定可比性。（市场上拥有数量众多的Lua相关书籍，如*Programming in Lua*（www.lua.org/pil/），这本书现在已出到第3版了。）尽管Squirrel语言相对默默无闻的状态可能会改变，但可能不会改变的事实是，Squirrel这个名字被用到太多地方了。除非你用的搜索引擎，如Google，能记录你的惯常行为，为你提供适合的文章，否则搜索诸如squirrel performance和squirrel numbers之类的关键词基本上得不到什么有用的结果。这种情况下，把language也放入关键字列表，或者把搜索区域限定为www.squirrel-lang.org/可能会得到有用的结果。

代码编写完成后，把代码推送到装置的过程体现了Electric Imp平台的一个重要优势。使用BlinkUp应用时，除了要给出WiFi连接的细节信息，还需要登录到electricimp.com账号。这意味着，在给Imp发送完信号后，Imp就知道自己关联到了哪个账号。当把Imp插入impee之后，impee所在的宿主装置会以节点的形式，显示在设计器（Planner）¹应用中（设计器位于electricimp.com网站上）。然后可以把该节点与所写的一段代码关联起来。在这之后，每当你把Imp插入impee，它就会连接到Web服务并把最新的代码下载下来。这意味着，在修改代码后，只需要把Imp弹出，然后再次插入，就能刷新Imp上的代码了。

¹ 根据electricimp.com网站的介绍，该平台已启用新的开发环境，设计器现已不推荐使用。
——译者注

直到这个时候，设计器应用才会显示任何可能存在的错误消息。这也展示了Squirrel语言的动态性。该语言是没有编译期检查的。

尽管部署环节做得很精致，但编辑器和设计器使用起来却有违常理。当然，随着平台的不断成熟，使用体验很可能会得到改善。来看一下让LED灯闪烁的例子：

```
// Blink-O-Matic 范例代码

local ledState = 0;

function blink()
{
    // 改变状态
    ledState = ledState?0:1;
    hardware.pin9.write(ledState);

    // 下一次状态的改变被安排到100ms后发生
    imp.wakeup(0.1, blink);
}

// 把引脚9配置为开漏输出模式，并启用内部的上拉电阻
hardware.pin9.configure(DIGITAL_OUT_OD_PULLUP);

// 向服务器注册
imp.configure("Blink-O-Matic", [], []);

// 开始闪烁
blink();
```

首先要注意的是，这段代码中没有像`setup()`和`loop()`函数那样分工明确的构成部分。所有和设置相关的功能都是在程序主体中实现的。`blink`函数被显式调用，并且会告知Imp在100毫秒后需再次调用该函数。除了这些之外，大多数对硬件引脚的常规配置，交替改变引脚的高低电平状态，这些代码看上去都比较熟悉。

这段代码的详细信息需要通过调用`imp.configure()`告诉设计器。我们只填写了一个参数，即代码的描述性名称。但如你所见，你还需

要提供两个额外的列表型的参数（在本例中，列表内容为空）。稍后我们会继续对此进行介绍。

• 调试

代码在部署之后或在运行期间产生的任何错误消息都能显示在编辑器窗口中。也可以在Imp上通过调用`server.show()`的方式输出消息。这个功能非常好用，和Arduino相比，这种实现方式要比串口控制台更容易使用，并且代码在运行过程中发生错误时，你也能自动获得有用的输出信息。

`server.show()`的输出也会被发送到设计器的界面，并被显示在impee对应的节点框上。当只有一个impee在运行时，这项功能给人的印象不是很深刻。当同时有若干个装置在运行时，通过节点框能让你对所有装置的状况有个较好的概貌性了解。如你现在所见，设计器界面对于真正有效使用Imp至关重要。

• 设计器

在设计器界面中在各个节点间建立连接，可能是该平台上最有意思的功能了。每个节点都可以输出数据，这些数据不仅可以显示在设计器界面上，也能输出到另一个节点。第二个节点需要有对应的输入接口，用来对数据做某种方式的处理。例如，一个Imp可以输出某开关的状态值，而另一个Imp则可以根据这个开关的状态来点亮或熄灭一盏灯。不管第二个Imp是在另一个房间里（与第一个Imp在同一个WiFi网络中），还是在不同的网络中（可能在另一个国家中），这个功能都能正常使用。你可以使用如下代码实现一个这样的系统：

```
// 节点1
local my_output = OutputPort("SwitchOut");
imp.configure("RemoteLightSwitch", [], [my_output]);
// 在适当的时候让灯点亮
my_output.set(1);

// 节点2
class SwitchIn extends InputPort
{
    type = "integer"
```

```
    name = "switch_value"
    function set(value) {
        hardware.pin1.write(value)
    }
}
imp.configure("Light", [Servo()], []);
```

Electric Imp平台会完全隐藏两个装置之间的通信细节，你不需要关注HTTP等协议的调用，只需要处理好在装置之间传送的消息。

你可能注意到，在上面这个例子中，调用`imp.configure`时，两个列表参数有适当的取值，而在上上个例子中，这两个列表全为空。这两个列表正是代码中实现网络通信的方式。第一个列表表示各种输入端口（从这里接收Electric Imp服务器发送给当前Imp的信息），而第二个列表给出了输出端口（当前Imp从这里发送数据到Electric Imp服务器）。当你定义好若干输入和输出端口后，你就能在设计器中通过连线的方式在它们之间建立连接，把一个Imp输出的信息注入到另一个Imp的输入端口。

因为实现Imp之间的通信不需要涉及HTTP通信等底层的各种细节，所以Imp不支持直接访问网络。如果故事到此为止，对于实现任何物联网产品而言，Imp显然是一个糟糕的选择。实际上，设计器应用允许你添加支持HTTP调用的节点。这些节点不是运行在Imp上，而是在云端，在`plan.electricimp.com`服务器上运行。考虑到Imp本身可能运行在网络流量受限的网络上，这种设计非常合理。服务器端的应用会负责处理网络调用相关的事项，并且把数据（例如JSON格式的数据）传送给下一个节点。

原本要在Arduino、树莓派或BeagleBone平台上，通过在单一微控制器上编写代码实现的功能，在Electric Imp平台上却可以分解到若干节点上实现。让我们通过以下几个例子，看看对于艾德里安和工作室中其他人制作过的一些物联网装置，怎样用Electric Imp实现它们的概念验证样机。

Bubblino

(1) 与Twitter服务器通信的节点需要驻留在Electric Imp服务器上，用来周期性地检查是否有新推文。

(2) 一旦发现有新推文，该节点就会与Imp进行通信，Imp会触发泡泡机工作，开始吹泡泡。

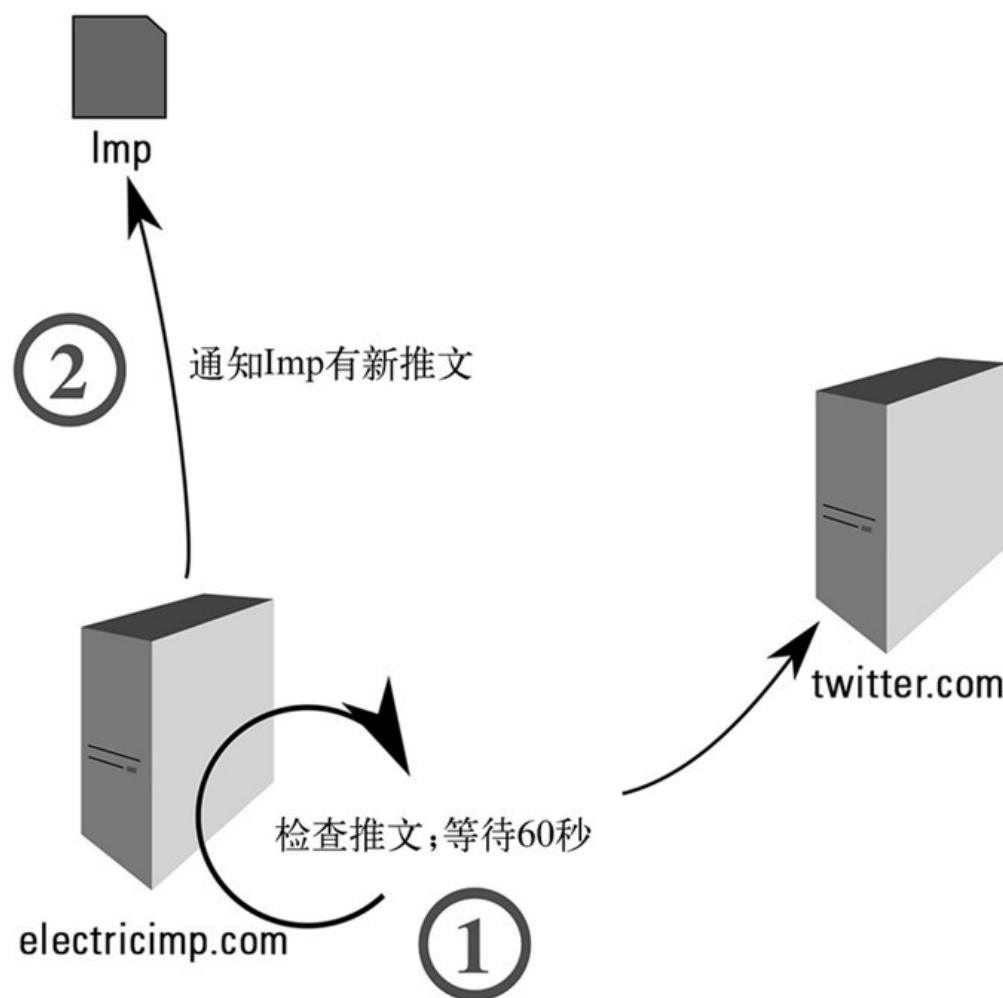


图 5-13 用Electric Imp实现Bubblino的网络拓扑图

晚安灯

这实际上也是我们在之前的章节中介绍过的例子。当大灯点亮时，它的Imp会经由Electric Imp服务器通知小灯的Imp。

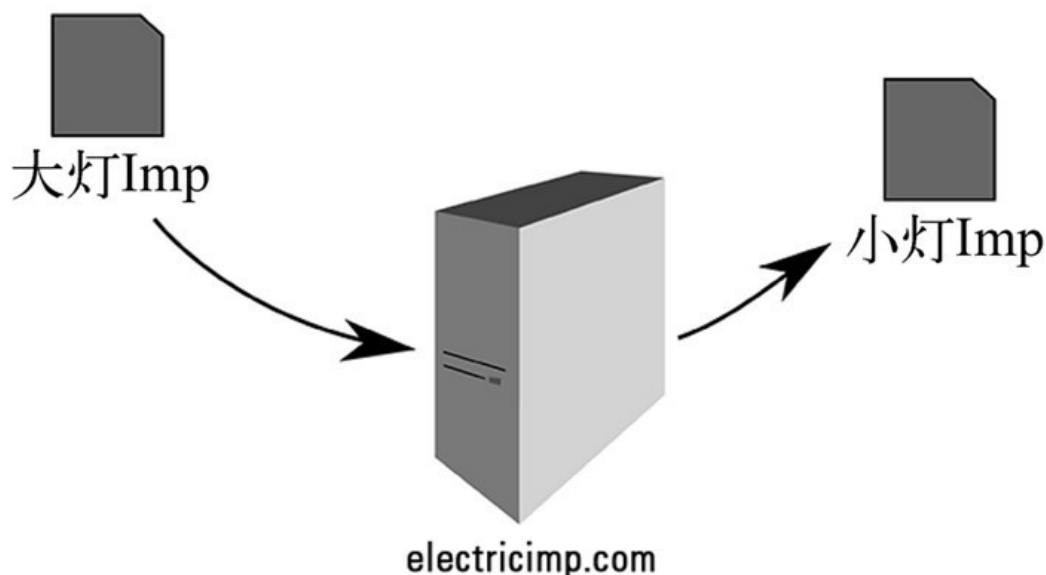


图 5-14 由单个大灯和小灯构成的晚安灯是Electric Imp平台上最简单的网络应用范例

YAHMS住宅管理系统

这套系统是我们用来控制办公室加热装置的系统。更多有关该系统工作过程的细节，请见第4章的案例研究：利物浦DoES。

(1) 办公室的温度监测可以用一个Imp实现，对应于下图中的监测Imp。这个Imp会定期地向位于Electric Imp的服务器上的一个节点报告温度，而该节点会再把这个温度值记录到Xively上的一个feed对象中。

(2) 独立于监测系统，位于Electric Imp的服务器上的一个控制节点会定期查询位于yahms.net的服务器，看是否需要打开或关闭加热装置，并且会通知到另一个Imp，即下图中的控制Imp。相应地，这个控制Imp会依照通知要求打开或关闭加热系统。

上述这几个例子都略去了配置和控制的细节信息。设计器应用允许对参数进行微调，但由于它太过灵活，很难把它集成到产品中。当然，Electric Imp在软件方面为制造商们提供了更多的选择，而不是只有这个面向开发阶段的设计器应用。

当前，我们只有为数不多的服务器节点或“虚拟impee”可用，但你能找到编写自己的服务器节点的相关文档（仍然采用Squirrel语言）。如果你有这方面的需求，我们鼓励你去联系Electric Imp团队。

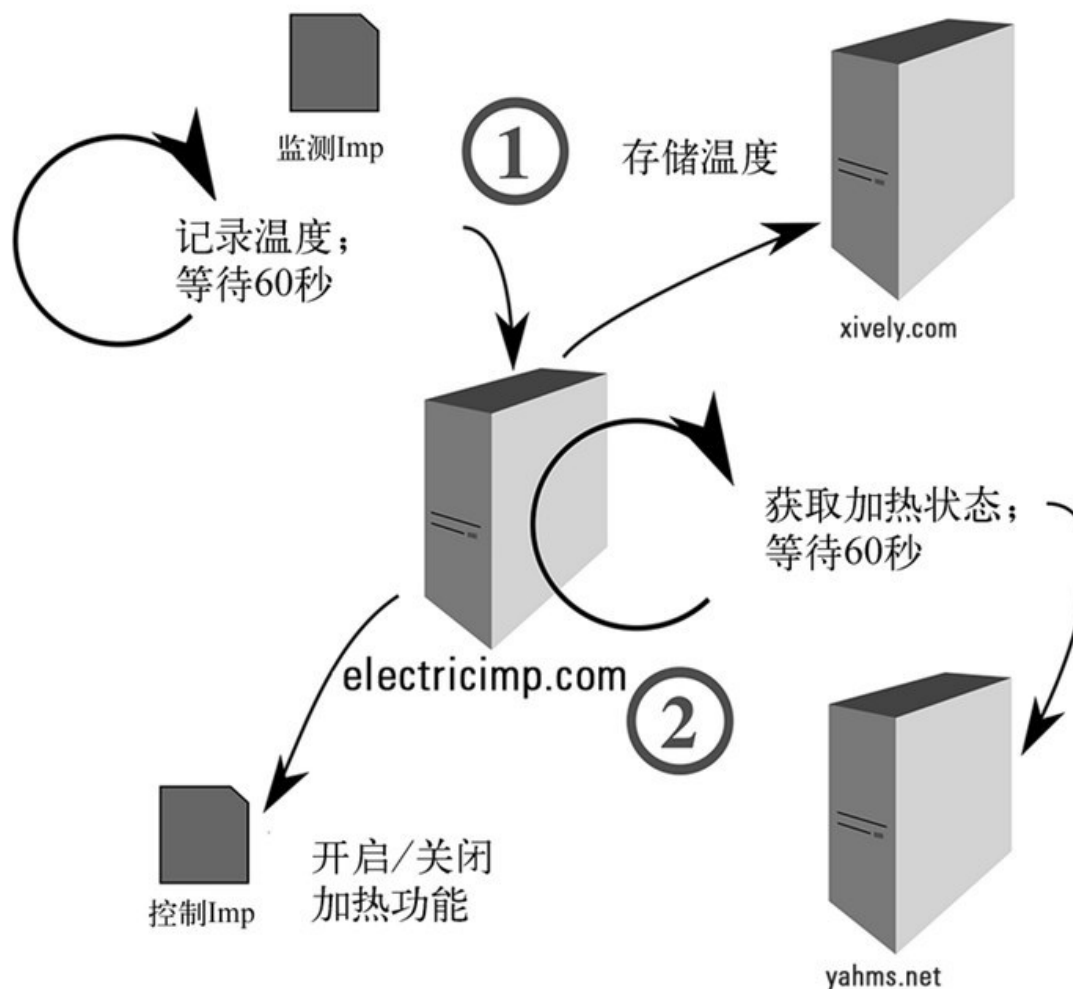


图 5-15 一个更复杂的Electric Imp应用场景，用于监测某个办公室的温度，并且允许对加热装置进行远程控制或自动控制

对组网方式的评论

采用这种透明的组网方式，你可以把服务器节点和一个或多个可能位于不同网络中的Imp连接起来，这非常有吸引力。但这也意味着在Electric Imp的服务器上，存在一个通信的中央节点，而该节点有可能会失效。采用这种组网方式也意味着，在同一个房间中的两个需要协同工作的Imp不能直接联网，而是要经由因特网

上的其他节点才能实现互相通信。这种网络配置方式使得Imp不适用于需要实现更快的mesh通信的场合。如果你想把通信范围扩展到WiFi网络之外的地方，例如，采用视距范围内的红外通信或远距离无线通信，在这种情况下Imp就缺乏吸引力了。

2. 开放性

对于本章中介绍的其他微控制器平台，你已经看到了相关商业生态系统是怎样成长的。Electric Imp不是为爱好者创建的，它的出现也不是出于教育目的，而是作为一个可以集成到产品中的商业系统出现的。同样，它没有针对小修小改的需求做优化，而是致力于把事情做成。一如所料，尽管该项目部分开源，但构成其竞争优势的两个主要因素——创新性的外形尺寸和在线设计器，目前还是坚决地采用了闭源方式。

• 外形尺寸

作为一个封装在SD卡塑料外壳中的小巧微控制器，Imp没有提供小修小改的能力。它不包含用户可自行维修的零部件，不提供指导你如何组装一个Imp的文档。

同样，在Imp上运行的操作系统也是保密的。它能运行BlinkUp应用，能使用起诊断测试作用的LED灯，能与impee通信并获取其身份标识，能与electricimp.com服务器进行代码的同步。然后，它能运行新的代码，并在适当的时候与服务器通信。没有用户可见的方式，用来改变这一工作过程。

• 设计器

设计器的源代码和用于连接Imp和虚拟节点的后端基础设施都是闭源的。这样做有原因的，毕竟这是一项商业服务。不过，这也意味着，如果选用Imp做开发，你就受到了限制，只能使用一个供应商提供的服务。你不得不信任一个第三方，让它确保你的代码是安全可用的，并已做好备份。这也不算是很大的改变。把业务过程的一部分，如主机服务（Heroku）、源代码控制（GitHub）、电子邮件发送（Postmark）和错误日志记录（Sentry），作为IT产品外包出去，是完全合理的做法。

然而，依赖于单一供应商的状况是一个现实的问题。不管前面几节中提到的供应商长期来看是不是最好的，它们都有切实可行的替代者。就**Electric Imp**而言，它在某种程度上是在做开创性的工作，把你的代码和装置在需要的时候转移到另一个服务提供者，可能没那么简单。你需要问自己这么几个问题：

- 如果它们关门了，有没有可能对**Imp**进行重新配置，使其指向另一个能托管相关代码的服务提供者？
- 如果它们提供的服务不够充分，你能做什么？
- 在你的项目做大之后，如果想对设计器之类的基础设施做一些配置，而**Electric Imp**团队却不想为普通客户提供这方面的便利，你怎么继续推进此事？它们会允许你赞助相关的开发吗？会让你自己管理这些基础设施吗？

• 编程语言

Squirrel编程语言的开发，实际上采用的是**MIT**开源许可。尽管和**Lua**等其他竞争者相比，它是一个相对较新的项目，但你还是能阅读**Squirrel**的源代码，为其修正错误和增加功能等。不过，在**Imp**和虚拟节点上运行的**Squirrel**语言的版本，是由**Electric Imp**团队全权负责维护的，对**Squirrel**的各种改变能否应用（或及时应用）到**Electric Imp**平台是无法保证的。

• Impee

impee的参考设计被放在了公共域

（<http://devwiki.electricimp.com/doku.php?id=boards:start>）。作为开发者，你在开始的时候很可能愿意使用小巧的**April**板，该板有6个引脚可用，能为**Imp**供电。尽管如此，你可以轻松地利用**Electric Imp**团队提供的原理图、材料清单和**Gerber**文件，创建自己的**impee**，使其与你的项目作品在外形尺寸上相匹配。（如果不熟悉上述术语，别担心，在第10章会有详细介绍。）

使用与**SD**卡相同的外形尺寸是一个极好的设计决策。为了能连接**Imp**，你可以购买现成的**SD**卡卡槽，把卡槽连接到一个廉价的

Atmel密码芯片（用来为impee提供唯一的身份标识），再加上几个其他的价格合理的元器件，你就得到了一个能用的用于连接Imp的宿主装置。

Electric Imp的主要目标用户群之一是现有的消费电子产品制造商。这些厂商想找到一种方便的方式，将物联网技术应用到产品中来。能容易地添加用于连接Imp的硬件装置，对他们而言是很有意义的。与把Arduino这样比较大的单板集成到产品中相比，在产品中找一块用于容纳SD卡卡槽和身份识别芯片的空间要容易得多。

Electric Imp案例研究：Lockitron

尽管基于树莓派平台的DoorBot为一群很懂技术的人解决了开门问题，但大多数人还是在用钥匙开门。Apigy公司的Lockitron (<https://lockitron.com/>) 提供了一个面向消费者的解决方案，在无需配制更多钥匙的前提下，解决了授权访问者进入房间的问题。同时，该方案不要求更换门锁，也无需在墙上打洞，而是用一个装置与当前门锁的旋钮适配，通过简单地带动旋钮转动实现开锁的目的。之后，可以用移动电话上的一个应用程序开锁，而非采用RFID方式（尽管Lockitron也提供了可选的对NFC的支持）。这意味着该装置必须一直连接因特网。由于很多人没有把以太网线缆布设到前门，这就要求具备无线联网能力。这种场合正是Electric Imp的用武之地。“WiFi接入的注册过程出了名的难，Electric Imp的BlinkUp应用承担了这项任务，并且把它转变为一件简单而愉快的事情。集成Electric Imp的过程简单得让人难以置信，我们无需为编写固件而去购买昂贵的软件包。” (<http://blog.electricimp.com/post/45920501558/lockitron>) 尽管Imp是我们介绍过的最简单的微控制器之一，但根据Lockitron特定的功能需求，只需要让Imp与锁做交互，感测锁是否已经打开，能把锁打开或锁上就可以了，所有其他的功能都在因特网中完成。因此，这是一个极好的案例，展示了Electric Imp的业务模型，即把任务分割后，由本地impee和云端节点分别完成。

5.7 其他值得关注的平台

正如上一章所述，现在已经存在很多原型制作平台了，将来还会有更多的平台出现。本章就未来数年中，物联网项目在选择平台时的可选方案，介绍了一些有根据的猜测。即便最终证明我们在平台选择方面判断有误，我们也希望对不同选择间的具体利弊分析能对你有所助益。

然而有几个平台，我们虽然不能对其做重点介绍，但也值得正式提及。现在就简要介绍下这些更有趣的可选方案吧。从低级的**Arduino**，到能力强大但又过分灵活的树莓派，再到封装更好、更“固执己见”的**BeagleBone**，最后到有吸引力但却封闭的**Electric Imp**，我们都一一作了介绍。接下来，再通过介绍一些硬件完全封闭的装置——移动电话、平板电脑和插头计算装置来继续这一进程。

不过，有一些方法可以用来迂回解决封闭性带来的问题，对安卓配件开发工具包（ADK）的介绍会说明这一点。

5.7.1 手机和平板电脑

新式的手机和平板电脑，不管它们搭载的是iOS系统（iPhone/iPad），还是安卓、黑莓或Windows系统，都配备了陀螺仪和温度计等传感器，装有一个或多个能捕捉静态图像和录制视频的摄像头、麦克风、GPS、WiFi、蓝牙、USB、NFC（非接触式银行卡或交通卡也是采用相同的技术）、按键以及支持多点触控的屏幕。手机通过WiFi或手机网络能提供持续的因特网连接。很多平板电脑也能提供同样的功能。这些装置还配备了若干种有吸引力的输出方式：高保真音频输出、高质量视频，以及一个或多个震动元件。它们的处理能力也与树莓派相当。（目前看来，它们做通用处理的速度可能比树莓派快。但由于树莓派的GPU异常强大，它们在图形处理方面会相对慢些。）

表面看来，使用移动终端创建物联网应用似乎是一种完美的解决方案。然而，这种诱惑往往要依赖于该装置作为电话的功能。在这种情况下，你实际是在为该装置创建一个应用，而不是在创建新的“物品”。尽管可以对移动电话的输入能力进行非常有创意地利用（例如，

通过分析皮肤颜色中红色成分的变化情况，使用摄像头测量脉搏），但把它连接到附属的电路或输入器件却不大容易实现。

最后，这些移动终端太大太贵了，不能被安装到任意的消费电子装置中。

安卓配件开发工具包

在制作原型时，成本和尺寸可能无关紧要，但不能连接附属的传感器和执行器则会妨碍你在正规的物联网项目中使用移动终端。安卓配件开发工具包（ADK）通过把一部安卓设备与一个兼容ADK的微控制器（如当前基于ARM的Arduino Due）配对使用解决了这个问题。我们看到移动设备缺乏与任意的电子器件通信的能力，但它们配备了USB接口，通常能用来与计算机同步数据。

在使用USB连接时，你通常需要区分USB主机（如计算机）和USB设备（如手机等）这两个角色。如果想把安卓设备连接到Arduino，后者需要承担主机的角色。实际使用时，安卓设备提供因特网连接、输入输出能力和处理能力，而Arduino本身仅提供访问其GPIO引脚的能力。

考虑到安卓设备和Arduino Due的成本和尺寸，以及其当前的文档状况，目前看来，ADK似乎还不是一个切实可行的平台。但在能连网的多媒体装置和物理计算平台之间分配任务的实现方式是未来一个有吸引力的方向。

5.7.2 插头计算：始终在线的物联网

不管计算装置做的有多小巧，你还是需要给它供电。即便是最小的单板也可能有一个相对比较厚重的电池组。插头计算装置的设计理念是把电路单元、插头和电源适配器集成在同一外壳中。事实证明这是一种非常便捷的外形设计，理由如下：

- 你不需要找地方来放置这个计算装置；
- 不会发生线缆被人移除或拔掉的情况；

- 装置能与插座紧密相连，不引人注目。

最后一项理由使得插头计算在包括安全专家和攻击者在内的一些群体中颇受欢迎。Pwn Plug (<http://pwnieexpress.com>) 是一个企业级的渗透测试工具，把它插入电源插座就能对典型的安全漏洞进行测试。尽管该产品宣称其用户是“白帽骇客”（即帮助公司确保数据安全的安全顾问），此类产品也能被用来做坏事。对于大多数雇员来说，一个插在墙上的崭新白色小盒子不会让他们觉得有什么特别，前提是他们真能注意到它的存在。

此类装置当然有合法的完美用途。例如，Freedom Box家庭办公室服务器 (http://p2pfoundation.net/Freedom_Box) 提供了一种廉价开源的解决方案，用来为小公司或家庭用户提供文件共享和电子邮件服务。

SheevaPlug (www.plugcomputer.org/) 是最流行的插头计算平台之一（如图5-16所示），通常运行在为其内部ARM芯片定制的Debian Linux之上，一般只配备以太网和USB接口。

我们已经介绍过的单板的功耗都比较小，能通过USB和小的电源适配器供电。因此，把树莓派或BeagleBone封装为一个插头计算装置很简单，并且该装置还能非常方便地访问GPIO引脚。SheevaPlug等产品未必是物联网产品的最佳解决方案，其吸引人的地方在于，它们已经是可买到的消费电子产品了。



图 5-16 SheevaPlug微型计算装置，看上去更像是一个电源适配器

5.8 小结

现在你已经开始能将大脑中相关项目需要用到的电子电路和嵌入式软件的概念结合到一起了。如果你在嵌入式开发方面还没有什么经验，不要太担心会做出“错误”的选择。在后续章节中你将看到，在物联网产品完工之前，在电子电路和软件方面还需要进行一定程度的返工。因此，马上就去选择一个最适合自己的平台吧，不要再反复思考哪个选择更加完美了。

Electric Imp有很大可能找到了一个不太容易被注意到的商机和市场定位，即托管的封闭系统。在此类系统上做开发很便捷，并且有商业实体负责代表其用户做正确的决策。用户在得到这些便利的同时，作为交换，也愿意放弃对一些细节的控制。在本书写作的时候，该平台给人的感觉还是不够完备，但它是一个很让人兴奋的新事物。该平台，或者与之类似的其他平台，很可能会成为将来一批物联网新产品的特征。

有充分理由认为，**Arduino**是一个已经得到认可、实际存在的可选平台。它在技术支持和文档的丰富程度方面没有竞争对手，它的开放性使得对其做扩展并安装到一个现有产品中的过程比较容易实现。采用**Arduino**的唯一不利之处是，它的功能有限。对于大多数的物理计算场景，这种简单性大有益处。但对于很多对安全性有一定要求的物联网应用来说，基础的**Arduino Uno**平台看上去有点能力不足了。

基于**Linux**的树莓派和**BeagleBone**能让你获得所需的所有处理能力和连通性，但却要付出复杂性增加的代价。如果你想把它们用到批量生产的产品中，单件产品的成本也会增加。如果只是考虑原型的成本，在树莓派和**BeagleBone Black**之间没什么好选的，影响你选择的更可能是一些其他因素。树莓派的知名度很高，相关的社区也比较成熟，而**BeagleBone Black**与电子器件接口的能力更强，也更容易演进到制造过程。相比之下，我们认为**BeagleBone Black**更有优势。

介绍完电子电路后，下一步就是介绍电子电路所属装置的结构形态。在下一章，我们将引领大家体验外形结构的设计过程，从最初的草

图，到能让设计成为实物的工具，都将予以介绍。同时，我们会对新的数字化制造技术，如激光切割和3D打印，给予特别的关注。

第6章 原型系统的结构与制作

正如第5章所述，物联网之所以是一个令人兴奋的领域，原因之一就在于它跨越了众多不同的学科，涉及软件、电子电路、用户体验设计和产品设计等诸多方面。

上面提到的最后一项，即产品设计，看似是很多初生的物联网项目的绊脚石。之所以有此判断，是因为我们看到，在Hackaday网站或Kickstarter网站上，大量的项目还只是一些裸露在外的电路板，或者只是把电路板随便找个手头有的盒子塞进去。如果想让物联网装置成功获得大众青睐，我们就需要改变这种设计缺失的现状。如果你是设计师，就给自己点个赞吧，我们需要更多像你这样的人参与进来；如果你不是设计师，也不要担心，本章将介绍一些帮助你在这方面入门的技术和工具。

6.1 准备工作

为了能对设计工作有充分的准备，你应该（在理想情况下）尽早开始，甚至在还没有什么特定的问题需要解决的时候就开始准备。培养对设计的兴趣并积累所喜欢的设计作品，这两者都需要持续不断地坚持。但幸运的是，如果你还没开始，那么现在就是最佳时机。更让人开心的是，这个过程看上去不像是在从事什么工作，甚至可能被你归类为有趣的事情。

所有需要你做的是培养对周边环境的兴趣，开始留意你所见到的大量的物理对象和所经历的各种体验。这个步骤有助于你搞清楚自己喜欢什么和不喜欢什么。一段时间之后，你就会清楚自己特别欣赏哪些特质。也许Colnago公路自行车的简洁而流畅的线条吸引着你，或者8英寸主厨刀在功能设计上的完美平衡令人钦佩，或者MacBook的MagSafe充电器扣接到电脑电源接口上的方式也会令你开心。所有这些观察都有助于你搞清楚，什么样的设计才是一个能被自己认可的优秀设计。

当然，你接触的地域越广泛，能为你所用的素材就越多。当然，我们可没有说，公司的会计会支付你海外度假的费用，但让自己接触一下不同的文化和环境，确实有助于保持一个人敏锐的判断力。不过，比较现实的选择可能是去偶尔参观一下画廊或博物馆，特别是去看看那些偏向设计或与设计相关的展览。你以前能知道**职业发展**会这么有趣吗？

随着准备工作的不断推进，可以通过拍照、在笔记本上做简略记录或画草图的方式，把能够启发你的事项的细节收集起来。然后再选择最适合自己的方式，把这些收集到的素材粘贴到一个老式的剪贴簿中，或者把它们保存到Tumblr、Pinterest或博客中。假以时日，你将构建出一个充满优秀设计的资料库，而这将有助于你在设计自己的物品时，激发出卓越的创造力。

另外还需注意，就（非常）基本的设计教育目标而言，一定要理解所用的工具，以及在真实世界（而不是数字世界）中制作物品过程中所体现出的**物质性**（materiality）。

在数字世界中制作物品时，可以专注于外观设计，物品可以呈现为任意的形状。而在真实世界中，却有着各种烦人的物理学定律以及制造工具的种种限制。宏伟的悬垂设计可能在3D设计软件中看上去很好，但这会使椅子的重心偏移，以至于无法让这种椅子站得稳。90°的直角设计也会限制你制作外壳的方式。例如，试图采用真空成型的方式制作，把模具移除几乎就不可能做到了，因为这样就会使注射成型工艺变得更复杂。

关于工具的说明

对于我们这里提到的一些工具，你之前可能没有任何相关的使用经验，这取决于你接受过的教育的程度（对某些工具而言，可能还取决于接受教育的早晚）。如果你不是工程师，很可能就真不知道有些工具的用处，或者何时使用它们。

我们将会在本章中对激光切割机、3D打印机和数控铣床予以正式介绍，但不会去详细介绍车床或真空成型机。

车床往往有两种主要形式，这取决于它们是被用于木材工件还是金属工件。不过，两种类型车床的工作方式是相同的：用卡盘把工件固定住，在电机的驱动下，卡盘带动工件沿某个轴旋转。然后让车刀与旋转的工件材料产生接触，从而把工件切削成期望的形状。围绕轴进行旋转的工作原理，必然使得那些旋转轴对称的工件能有比较好的加工效果。

车床的工作方式和钻孔机有点像，但是要反过来看。钻孔机工作时，一个旋转的钻头与静止的工件产生接触，而车床工作时，却是让静止不动的刀头与旋转的工件产生接触。

通过把塑料片材吸附到模具上的方式，真空成型能够以一种简单的方法制作轻薄的塑料外壳。塑料片材在一个框架内被加热到非常软的可塑状态，然后把与成品外形相符的模具推入已经软化的塑料，并且利用真空力使塑料紧紧的吸附在模具上。这种技术不适用于有很多细节或非常深的模具，但一旦做好模具，就能非常快地大量生产同类的工件，即便是对于需要人工操作的机器来说，也是如此。

和大多数事情一样，在不同的材料和工具方面，积累的经验越多，做设计的时候就会对要做的工作有更多理解，所以，基于这个理由，你更应该主动搜寻并加入本地区的创意空间或创客空间。这些地方大都拥有一些没必要独立购买的工具，例如，激光切割机或3D打印机。很多地方还有数控铣床、车床或真空成型机等其他设备。在这些地方，很多专家都会给予人们丰富的建议和意见，而作为社区成员，你也完全可以利用这一有利条件。

因此，在等待灵感闪现的过程中，开始把玩手头的工具吧。参与到本地（或稍远一点的地方）的创意日活动中，制作一些物品并把它们作为礼物送出去（或仅仅是看看会发生什么）。例如，在过去的一个月里，利物浦DoES（我们帮助创立的创客空间）的成员采用激光切割的方式用桦木胶合板切割出一个比例缩小的长毛猛犸象的骨架，用3D打印机制作了一枚丢失的国际象棋棋子，还用激光切割机试着把一些毛毡切割成机器人的形状。

6.2 画草图，迭代和探索

即便做了很多的准备工作，当你最终坐下来考虑一个特定的项目时，你还是要更多地搜集和探索可能的着手点。可以认为，在设计开始阶段，无论对想法迭代多少次都不算多，你需要尝试用不同的方法来解决问题。

新的数字制造工具，如3D打印机和数控加工设备，被用来生产数量不多（小批量或单件）但看起来还挺专业的产品。尽管本章用了较多篇幅介绍这些新事物，但我们打算先介绍一些科技程度不太高的解决方案。

你可能一坐下来做设计，就会禁不住启动3D设计软件。然而，最初的想法很可能不是最好的，因此需要优化迭代的速度，而不是原型的质量。你是可以用3D打印机对各种设计方案进行逐一验证，但用笔纸无疑要快得多。

首先，你需要对问题域进行广泛搜索。这样做的目的是尽可能对设计的各个方面都有所把握，而不是专门对一个特定的可能解决方案进行深入钻研。

不要把自己局限于明显的解决方案，你必须以不同的方式来看待事物，从而更有可能在若干选项中找到优秀设计。

使用任何最可行的有助于想法的产生和探索的工具。你可以使用情绪板（**mood board**），在若干天内快速记下想法并绘制草图，或者使用纸质笔记本，当你坐在公园长椅上时，信手涂鸦式地画一些草图。

一些读者可能现在要抗议了：“这些做法都很好，但我不会画图。”这其实不算什么大问题。这些草图不是要放到美术馆的，它们只是用来帮助你厘清自己的思路，它们只要能捕捉和传递想法就可以了。画的草图越多，这些草图的质量也就越好。不过说到底，如果只会画简笔画，也不要担心，继续画你的小人儿就好。

如果觉得有必要（甚至可能是没必要）进一步验证某个想法，要敢于把草图实现为三维模型。用雕塑粘土、乐高积木或本章介绍的其他方

法，把不同的设计想法实现为实体模型。尝试一下不同的尺寸，看看尺寸的改变会怎样影响设计的感觉或外观。

也许可以把这里介绍的方法和第5章中有关原型系统的软件和电子电路设计的内容结合起来，快速制作一个粗糙但还可以用的原型，以便能做适当的尝试。然后把它交给一些可能会使用最终产品的人，或把原型展示给他们看，看看他们是怎么与之交互的。

关键的一点是使用这些技术尝试不同的可能性，了解哪种设计的哪个功能是最好的。这种方式允许你把不同探索的结果综合为一种清晰明确的最终设计。

作为一个例子，让我们看一下晚安灯设计的演进过程（关于晚安灯的具体介绍，请参看第5章）。最初设计出的形状是一种比较传统的台灯，但在一次设计研讨会上，借助于一个纯粹的功能原型，设计团队就一系列想法进行了讨论。他们意识到，模仿房子外形的设计能更好地传达出“把亲人联系起来”这一核心概念。

研讨会结束后，先是按照新的设计，用激光对亚克力和胶合板进行切割，制作了实体模型；然后用数控铣床对亚克力和中密度纤维板进行加工，制作了功能原型。



图 6-1 晚安灯的设计演进过程（从左到右）：最初的设计；功能性的实体模型；重新设计的实体模型；重新设计的功能原型（橙色和木料/亚克力）；尺寸调整后的实体模型；尺寸调整后的功能原型

对于这个新的设计，几周之后，他们就意识到这个原型稍微有点大了，于是又制作了更多的实体模型。这次是采用手工切割胶合板的方式，对不同的尺寸进行了尝试。选好尺寸后，他们制作（用数控铣床）和组装出了新的功能原型。修改尺寸后的功能原型被证明要比之前好很多，从而成为了第一批产品的设计基础。

既然已经介绍了一些用来完成你的设计的比较通用的方法，我们可以对实现原型的一些特定的技术进行更深入的探讨。

6.3 非数字化的方法

我们已经看到，笔和纸仍然是设计师的工具库中必不可少的工具，但它们不是数字化变革后仅有的幸存者。很多被认为是更为传统的工艺技术，在制作装置原型的外形结构时依然有用。

与新的数字化制造方法相比，这些传统技术的一个关键优势就是即时性。3D打印的时间经常要以小时为单位计量，尽管激光切割要快很多，但切割一次仍然需要花费数分钟时间。这还都没有包括在计算机上修改设计的时间。

相比之下，对一个用粘土或乐高积木制作的模型进行重新配置要快得多——当然，在你成长过程中投入的练习时间没有计算在内。让产生想法和验证想法之间的反馈环尽可能短，你就可以有更多时间做更多的尝试。

让我们在这里介绍几种常见的传统技术。

- **雕塑粘土**：最知名的品牌是培乐多（Play-Doh）和普莱斯蒂辛（Plasticine），但你能找到很多质量略有差异的同类产品。一些产品，如培乐多，被暴露在空气中时，比较容易变干和开裂。Plasticine橡皮泥则不存在这个问题，但它会一直保持可塑性，对于需要操控的原型就不太适合了。雕塑粘土最适宜用来在短期内对形状进行探索，而不适合用来制作长期使用的功能原型。
- **环氧腻子**：你可能已经见到过Milliput品牌的此类产品。它类似于雕塑粘土，不过通常可供选择的颜色比较少。它一般由两部分组成，其一为固化剂。把二部分按等量混合起来，使环氧树脂活化，之后你就可以把它做成期望的形状。大约一个小时过后，它就凝固了。如果你愿意做得更好，还可以进一步对它进行打磨或涂色。由此可见，此类产品适合制作需要长久保存的物品。
- **Sugru**：Sugru，也叫万能粘土，是一种可塑硅胶。和环氧腻子一样，只能在短时间内改变其造型（大约30分钟左右开始凝结，过一天左右完全凝固）。但不同于环氧腻子的是，它在凝固之后仍

具有弹性。Sugru还非常容易粘在其他物质上，形成触感柔软且方便抓持的表面，这使得它成为了设计师（和黑客）工具集的一个很好的补充。

- **拼装玩具：** 我们已经提到过随处可见的乐高套件，但你也可以考虑Meccano（在美国市场使用Erector品牌）和很多其他的选择。如果运气好的话，你已经可以在阁楼里找到一些尘封已久的此类玩具，或者可以向你的孩子们借用。此类玩具的一个有趣之处是，它们包括了齿轮、铰链等组件，可以用来为模型增加运动能力。你可以购买专门的系统，用计算机控制乐高套件，但你其实没必要这么做。很多黑客用Arduino实现感测和控制，同时用乐高套件实现外形和连接装置，这种构建原型的方式兼具灵活性好和实现难度低的优点。
- **硬纸板：** 硬纸板价格便宜，易于用美工刀或剪刀制作造型，并且在颜色和厚度方面可以有各种选择。瓦楞纸板能基本实现完整的结构，能较好地勾勒出原型的外形。这个外形和以后用激光切割机把薄胶合板或亚克力板加工成型的结果是一致的（本章后续部分介绍激光切割机时，我们再继续这一话题）。
- **纸面泡沫板：** 这种板材由两层纸板和夹在纸板之间的泡沫材料层构成，可以在美术用品商店里买到，有3mm和5mm厚等一系列可选的厚度尺寸。和硬纸板一样，用美工刀就能很容易地对泡沫板进行切割。但与瓦楞纸板相比，纸面泡沫板有更好的刚性。还有专门用于泡沫板切割的美工刀，可以用来很容易地切割45°的斜面。有些美工刀还具备间隔3mm的两个刀片，可以很容易切割出插槽。这样你就能把另一个泡沫板插入其中，形成立体的形状。
登录
http://www.paulos.net/teaching/2011/BID/assignments/Foamcore_construction.pdf，可以看到一个很不错的用泡沫板制作物品的入门指引。
- **挤塑聚苯乙烯板（挤塑板）：** 本产品类似于用于包装的发泡聚苯乙烯板，但这种泡沫体的密度可要大得多，因此更适于建模。尽管重要的是密度而不是颜色，它还是经常被人称为“蓝泡沫塑料”（blue foam）。这种材料重量轻，经久耐用，且易于加工：你

可以用美工刀切割，用锯子锯，用打磨机打磨。或者，为了能更方便地对其进行造型，还可以买一台热线切割机。挤塑板比泡沫板要厚很多，通常厚度在25毫米到165毫米之间，所以很适合制作实心的三维模型。如果需要的厚度大于板材本身的厚度，可以很容易地把几层板材粘合到一起。注意：用打磨机打磨时产生的粉尘，或者用热线切割机切割时产生的烟雾对人体是有害的，所以要确保在做这些事之前佩戴防尘口罩，并且要在工作场所保持良好的通风环境。

回顾了小学时代在设计教育启蒙阶段学到的各种技术之后，我们可以继续介绍一些新的工具。与现代生活的大多数领域一样，计算机已经横扫制造过程，为快速原型设计开创出新的可能性。摩尔定律推动了计算成本的下降，20世纪80年代的早期开发活动中形成的专利也已逐渐过期，在这两个因素的共同作用下，爱好者或小企业也有能力获得这些技术了。

6.4 激光切割

虽然激光切割机不像3D打印机那样受媒体关注，但在你的工作坊中，它可以说是一个更加有用的工具。虽然能用3D打印机制作较复杂的零部件，但对于很多形状复杂的零部件，把它们分解为一系列二维的平面形状的零部件并分别进行设计，要比直接进行三维设计更容易一些。激光切割机对应的设计过程较简单，可切割材料的选择范围较大，并且切割速度也较快，这使得它成为一种多用途工具。

激光切割机的种类众多，它既可以是小型的桌面式装置，也可以是大型的工业设备。大型激光切割机可以对8英尺×4英尺幅面的板材一次性地完成切割。不过，最常见的激光切割机是落地式的，其体积和一台大型复印机相当。

激光切割机中最大的一个部件是工作台，这是一个用来放置被切割材料的平坦面板。工作台上有一个二轴定位机构，它配备有数个反射镜和一个透镜，用来把激光束投射到正确的位置并聚焦于被切割材料上。虽然在外型上与平板绘图仪相似，但激光切割机是对材料进行灼烧，而不是在材料上绘制图形。

二轴定位机构的位置和激光束的功率都是由计算机控制的。这意味着激光切割机不仅可以毫不费力地切割出各种复杂的图案，也可以通过降低激光输出功率的方式不把板材完全切穿。在激光输出功率足够小的情况下，这个特性允许你在工件的表面蚀刻一些额外的细节。你还可以在不同的功率级别下进行蚀刻，从而获得不同的蚀刻深度。激光切割机目前还做不到把蚀刻深度设定为零点几毫米，尽管这种深度应该是肉眼可辨识的。

6.4.1 激光切割机的选择

选择激光切割机时，应该考虑两个主要的特征。

- **工作台尺寸：**工作台是切割时放置板材的地方，较大的工作台可以用来切割较大尺寸的板材。你不要只考虑自己创建的物品最大能有多大，较大的工作台使得你可以购买较大尺寸的板材（这样

也比较划算)。此外,如果你进入小规模生产阶段,较大的工作台使得你可以一次切割多个部件。

- **激光器的功率**: 激光器的输出功率越高,能切割的板材就越厚。例如,作者车间里的激光切割机,它的激光器的输出功率为40W,它可以用来切割最多10mm厚的亚克力板。如果使用同系列的配备了60W激光器的激光切割机,就可以切割25mm厚的亚克力板。

你可以用激光切割机切割各种不同材料,这取决于你想制作什么。虽然毛毡、皮革和其他各种纺织品易于切割,但对于物联网装置,你可能会考虑使用硬一点的材料。纸板,特别是瓦楞纸板,适合用来做快速测试和制作原型,但中密度纤维板、胶合板和亚克力板(根据Perspex品牌的名称,也俗称珀斯佩有机玻璃)是最常见的选择。

还有一些特殊的材料可供选用,用来实现特定的目的。例如,对于能用激光蚀刻的橡胶材料,可以用它来制作印章。再例如,亚克力层压板是由一种颜色的表面薄板和颜色与之对比强烈的底层厚板压合而成,因此可以通过对薄板进行蚀刻,形成清晰分明和高对比度的细节和文字。

尽管你能够获得可切割金属的激光切割机,但它们往往是大功率的工业装备。低功率的型号不仅不能切穿金属,更糟糕的是,很多金属板材具有的光亮表面能很好地反射激光束,你还很有可能损坏机器。激光切割机可用于蚀刻金属,但前提是已经预先用陶瓷复合涂层(如CerMark)对反射表面进行了精心覆盖。无论是用喷雾罐还是用胶带,一旦形成涂层,激光就会使复合涂层及其下面的金属融合,留下永久的黑色印记。

如果没有自己的激光切割机,所在地的创客空间或创意空间很有可能会提供。你甚至还可能获得本地大学的许可,使用他们的机器。如果这些途径都没有,还可以去找专门提供激光切割服务的机构,它们有点像复印店,现在变得越来越常见了。建筑师们会经常利用这些服务,用来帮助他们构建建筑模型,因此你可以先找找这种机构。如果这个办法也不可行,还可以寻求一些在线服务商的支持,如Ponoko

(<http://www.ponoko.com>)，在接收到你上传的相关设计文件后，他们会把切割好的零部件邮寄给你。

6.4.2 软件

不同的机器和服务提供商，对用来呈现设计的软件及文件格式的要求也不一样。尽管一些激光切割软件允许用位图表示蚀刻图案，但通常用的是矢量图。

矢量格式用一系列的直线和曲线来描述图形。和位图网格状的表示方式相比，矢量图能更容易地转换为激光切割机中的移动指令。同时，调整图像尺寸也不会产生失真。使用位图时，你可能在尝试把数码相片的一小部分放大时已经看到，随着图像的放大，细节部分变成了锯齿状。而矢量图知道被放大的地方还是一根直线，因此能够在重绘时保留细节。

CorelDRAW是利用激光切割机工作时的常用软件，你也能用它来进行设计。其他流行的软件包括：Adobe Illustrator，很多设计师都在用这个软件，并且十分熟悉其用法；Inkscape，人们之所以选择它，很大程度上是因为这是一个可以免费获取的开源软件。对于你来说，哪个软件用得最顺手，哪个就是最佳选择。如果没有熟悉的软件，可以选择激光切割机现在使用的软件，或者选择一个买得起的软件。

设计时，需要使用形状和线条的笔画（或外轮廓线）——而非它们内部的填充区域——来定义激光切割和蚀刻的位置。激光切割时产生的切缝大约是0.2毫米宽，你不需要在设计中考虑这个宽度。使用较细的笔画宽度比较好，可以避免激光切割机产生误解，错误地在仅需要切割一次的地方做两次切割。

不同类型的操作，如切割、蚀刻，或者不同级别的蚀刻，通常可以放到同一个设计文件中，标记上不同的颜色加以区分。为你提供切割服务的地方，对于各种颜色和切割参数的对应关系，应该有一个明确的规范。如果确实有这样的规范，你就应该要遵守它。

6.4.3 铰链和接头

用激光切割机加工的用来构造物品的大多数机械构件，和常见的木制品中的构件没什么不同。然而，有几个相对不太知名的技术有必要介绍一下。它们或者因为激光切割的精确性而变得容易制作出来，或者因为受到新一代创客的青睐而重新流行起来。

格栅式铰链（活动铰链）

如果你想在自己的设计中引入一些曲面，请先回想一下水果馅饼上面栅格状的酥皮，在这些栅格状的图案中选一种，就能达成设想。在与曲面垂直的方向上，进行一系列密集排列的切割，就能使板材在切割后可以被弯曲。改变切割的次数和切割线之间的间隔，会影响铰链的柔韧性。Patrick Fenner有一篇有趣的博文

（<http://www.deferredprocrastination.co.uk/blog/2011/laser-cut-lattice-living-hinges/>），深入探讨了不同的参数怎样影响最终的结果。

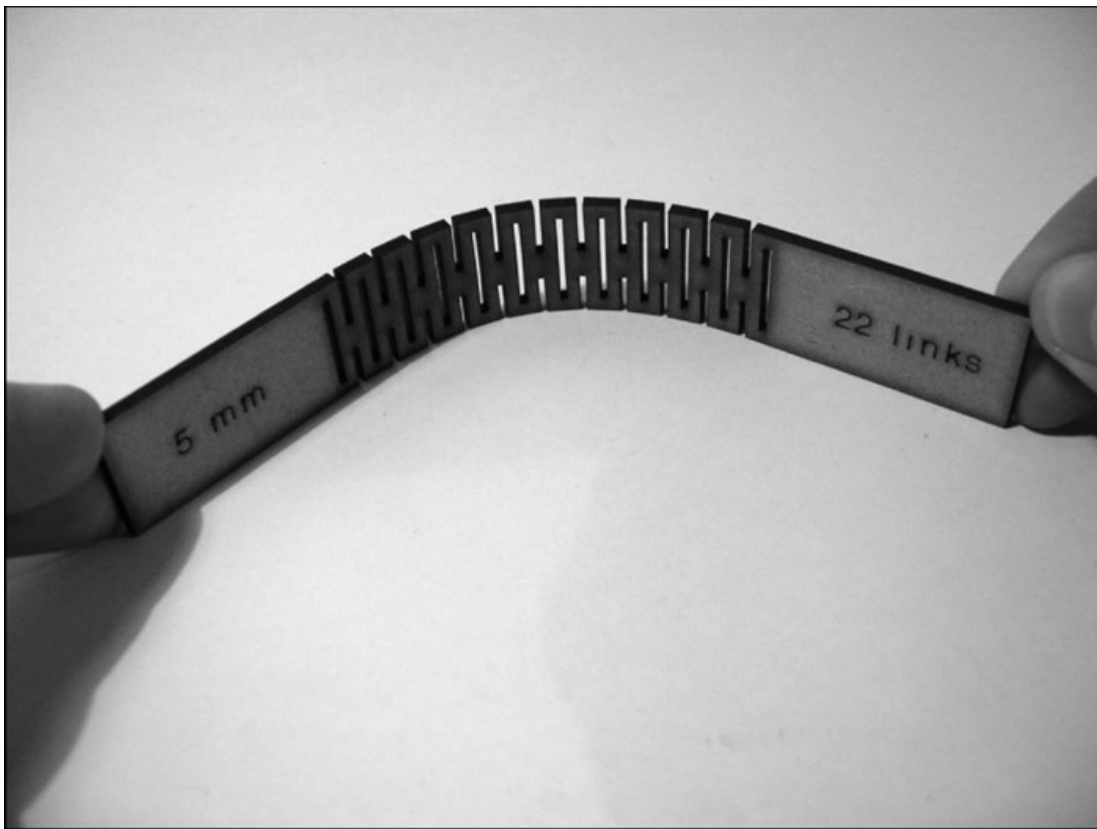


图 6-2 格栅式铰链（活动铰链）

一体化弹性夹

这种接头技术使用的场合和通榫接头相似，即需要把两片板件以90度角接合时。榫头（榫舌）被实现为两个挂钩的形式，一个挂钩向上突起，另一个挂钩的突起方向与榫眼所在板件的边沿平行，这样就能使榫眼和榫头所在的板件紧密接合，无需使用胶水或额外的固定装置。如图6-3所示，为了能为榫头提供所需的弹性，使其能装配到榫眼中，在榫头一侧做了一些额外的较深的切割。Patrick Fenner在这个领域也做了探索，相关的博文请见：<http://www.def-proc.co.uk/b/category/def-proc/laser-cut-elastic-clips/>。

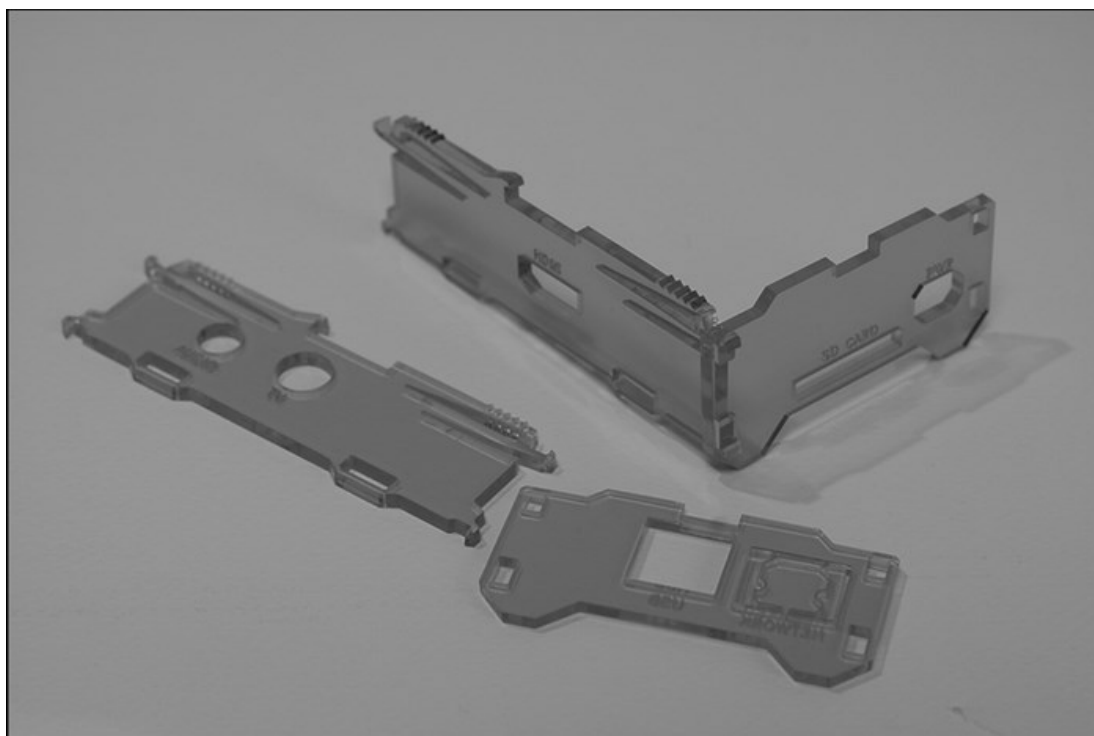


图 6-3 一体化弹性夹

用螺栓固定的榫接头（T型槽接头）

一体化弹性夹的一个替代品是用螺栓固定的榫接头。这种接头对标准的榫接头进行了修改，在榫头所在的板件上增加了一个T型或十字型的槽，这个槽的大小正好能容纳一个螺栓。然后你就可以把螺栓穿过榫眼所在板件上的小孔，再穿过在这个槽里放置的螺母。



图 6-4 用螺栓固定的榫接头（T型槽接头）

案例研究：Nick O'Leary的环境感应球

IBM的开发人员Nick O'Leary利用业余时间制作了一个支持多通道、多种颜色的环境感应球，用来帮助他留意诸如家庭能源使用情况之类的事情。

IBM开发的消息队列遥测传输（MQTT）协议被用来控制感应球。该协议能轻松把输出方（订阅者）连接到数据源（发布者）。感应球中有三个可以独立控制的RGB LED灯，它们分别与Nick所使用的MQTT即时通讯系统的不同订阅有着映射关系。例如，如果他注意到放置在厨房中的感应球的一侧正在发红光，其原因可能是他忘记关闭办公室的暖风机了。

为了能制作成一个适合放在家里的比较美观的物品，光有一块裸露在外的电路板是不行的，Nick想要一个好点的设计。他把一个从商店买来的灯具稍加修改，用作扩散光线的球体。但感应球的基座部分需要容纳一块完全不同的电路板，因此他要自己设计一个新的基座。

他想用木头制作这个基座，因此3D打印就排除在外了。这使得他转向了基于激光切割的设计，并通过把多层桦木胶合板堆叠在一起的方式获得需要的高度。

因为Nick手头没有激光切割机，他最初的设计是手工制作完成的，是用美工刀在瓦楞纸板上切割出来的。通过这个过程，Nick弄明白了，为了容纳电路板、电源连接器等构件，为了提供一个把基座的各部分固定到一起的机制，怎样对各层进行布局。这种固定机制也意味着，将来他可以对基座进行拆卸，以便进行任何的维护或升级。



图 6-5 环境感应球的基座

当Nick对设计满意后，他就其他地方利用激光切割完成了加工。你可以看到图6-5最右侧的是纸板做的原型，左边的是用激光切割的两个不同版本的桦木胶合板基座。因为发现留的间隙不够大，不能方便地放置PCB，所以他又重新做了设计，最终做出了一个很不错的、比较专业的设计成果。

访问网址<http://knolleary.net/orb/>，可以获得更多有关构建过程的资源，包括源代码和PCB设计。

6.5 3D打印

增材制造，也常被称为3D打印，正迅速成为最流行的、快速制作原型的方式之一。这主要归因于个人3D打印机的数量在不断增长，而其价格却在一直下降。现在一些桌面级的机型，价格还不到500英镑，但打印质量已经算不错了。

之所以采用增材制造这个术语，是因为各种用来产生输出的工艺流程都是采用从无到有，逐渐添加材料的方式来构建出最终的模型。这和诸如激光切割和数控铣之类的“减法式”制造技术形成了对比，后者都是在开始的时候有一块较大的原材料，然后把不需要的部分切割掉。

构建物理模型时使用的各种工艺流程，以及一些其他因素，都会对打印机能使用的打印材料造成影响。然而，所有3D打印机都是把计算机三维模型作为其输入。打印软件会把计算机三维模型切割为很多层，每层的厚度都是零点几毫米，从而一层层地构建对应的物理模型。

3D打印一个最吸引人的地方在于它可以用来生产传统技术无法制造的物品。例如，由于你能用3D打印机打印出没有任何接头的两个套在一起的圆环，所以你也可以用金属3D打印机打印整片的锁子甲。从打印机中出来时，锁子甲已经连为一体了。要是中世纪的骑士们能使用激光金属粉末烧结机的话，他们盔甲的制作就要容易得多了。

3D打印的另一个常用技巧是打印包含可移动部件的工件。也就是说，同时打印所有的部件，打印出来的就是一个组装好的工件。这种免装配的效果是通过使用所谓的“支撑材料”实现的。采用某些类型的打印技术，如粉末成型法时，免装配的效果只是此类打印技术的副效应。打印过程中，粉末状的原料会占据被打印物品的空气间隙。完成打印后，只需通过抖动或吹拂的方式去掉被打印物品上的松散粉末即可。另外一些打印技术，如挤塑类技术，需要你用另一种材料打印支撑体。完成打印后，支撑材料可以被折断或冲洗掉（支撑材料经过专门选择，能溶于水或某种溶液，并且该溶液不会对打印材料造成任何影响）。

6.5.1 3D打印技术的类型

在增材制造领域中，很多创新技术仍在不断涌现。下面列出的是目前最常用的几种3D打印技术。

- **熔丝制造（FFF）**：也被称为熔融沉积成型（FDM），你最有可能在创客们的聚会活动现场看到此类3D打印机。RepRap和MakerBot都采用了这种技术，而Stratasys在其工业级产品中也采用该技术。此类打印机工作时，细丝状的打印材料（一般是塑料）被送入机器，再从一个被加热的喷嘴挤出。在计算机的控制下，喷嘴可以沿水平方向和垂直方向移动，与此同时，熔融的细丝从喷嘴中流出并流到当前的位置。用这种方式打印出的模型非常牢固，因为它们就是用普通的塑料制成的。不过，因为细丝状的打印材料有一定的厚度，这些模型的表面看上去会较为粗糙。
- **激光烧结**：这种工艺流程有时被称为选择性激光烧结（SLS）、电子束熔融（EBM）或直接金属激光烧结（DMLS）。该技术被较多地用于工业级设备上，能用来打印任何粉末形式的，能被激光熔融的材料。和FDM相比，打印出来的成品表面更为精细，而牢固程度却没有变化，甚至会变得更为坚固（当打印介质为金属时）。该技术可被用于打印铝或钛制品，不过它也可以很容易地打印尼龙制品。MakieLab（将在第9章中介绍）使用激光烧结技术3D打印尼龙材质的实物玩偶。
- **粉末床**：激光烧结和粉末床都是使用粉末状的原材料，但前者用激光把粉末熔接在一起，而后者用粘合剂把粉末粘到一起。这种粘合剂更像是胶水，由一个打印头（类似于喷墨打印机的打印头）控制它的流出量。Z Corp.公司的3D打印机采用了这种技术，并且使用了一种类似熟石膏的材质作为打印介质。刚打印出来的模型非常易碎，需要进行后期处理，给它们喷洒硬化剂。这种打印机的最大优点是，在喷涂粘合剂的时候，可以在其中混合某种颜料，这样就能一次性地打印出色彩丰富的彩色模型。
- **分层实体制造（LOM）**：这是另外一种能生成彩色模型的方法。LOM把传统的纸张打印作为其工艺流程的一部分。因为这种技术构建模型的方式是把多层纸张粘合在一起，把纸张切割成型并粘接到已成型部分之前，可以在每层纸上打印任何需要的颜色。Mcor IRIS就是此类打印机中的一员。

- **立体光固化成型和数字光处理：** 立体光固化成型可能是最早出现的3D打印技术，和数字光处理（DLP）有很多共同点，而后者在本书写作时正处于人气飙升的状态。这两种方法都是用一罐液态树脂来构建模型，通过用紫外线照射的方式使树脂固化。立体光固化成型使用紫外激光器绘出每一层的图案，而数字光处理使用DLP投影仪，能快速地固化整个一层。虽然这两种方式只能使用树脂作为打印介质，但能生成非常精细的模型。较高的精细度，加之DLP投影仪相对成本较低，使得数字光处理在开发更为廉价的高精细度打印机时，成为一个最值得进行投入的领域。

决定使用哪一种3D打印机，可能主要是看你有机会获得哪种类型的设备。工业级设备的价格高达数万英镑，大多数人不会花费这么多钱去买它的。如果你的确有那么多钱去买这种设备，我们只能表示羡慕和妒忌了。祝你购物愉快！

大多数人有以下几种选择方案。如果你住在某个微观装配实验室（Fab Lab）或TechShop工作室附近，它们通常都有3D打印机，并且允许你使用。同样，本地大学的工程系或产品设计系通常也拥有此类设施，并可能会向你开放使用权限。

另外，或许也能找到一个本地的打印服务提供商，可以在那里把你的设计打印出来。此类服务越来越常见了。史泰博（Staple）公司最近就宣布了一项服务，顾客可以在该公司位于荷兰境内的门店中领取打印好的物品。

史泰博公司的公告尽管获得了较高的知晓度，但它只是进入“3D打印邮政服务”市场的新成员之一。Shapeways（<http://www.shapeways.com/>）、i.materialise（<http://i.materialise.com/>）和Ponoko（<https://www.ponoko.com/>）已经提供类似服务并运作一段时间了。你需要在线上传设计方案，选择打印方式，几天之后就能在邮筒中收到实物。很多服务商甚至会帮你售卖自己的设计作品，并连带处理销售过程中的各类事项。

如果不需要高端3D打印机的特性，如能使用特殊材料，或者较高的精细度，那么很有可能会在本地的创意空间或创客空间找到一台低成本的桌面型3D打印机。这些低端的3D打印机很廉价，当然也可以买一台

自用。实际上，对于大多数的原型制作工作，选择方便获得的设备和成本较低的打印方式是远远利大于弊的。

图6-6展示了几个用常见的桌面型3D打印机打印的样品。作为对比，把它们和一个用高端的Z Corp.打印机打印的样品放在了一起。



图 6-6 几个3D打印的样品（从左到右）：MakerBot Replicator、Z Corp.、Ultimaker、RepRap Prusa Mendel

现如今有大量的桌面型3D打印机可供选择，这主要归功于RepRap项目。该项目是由Adrian Bowyer在2004年的时候发起的，旨在构建一个能够自我复制的用于快速原型制作的设备。虽然他们迄今还未能实现用该设备打印电机之类的装置，但已经能把其自身一半的部件打印出来。剩下的另一半零部件则可以从硬件商店方便地获得，或者可以在线购买。

RepRap的设计是完全开源的，这意味着用户可以自由地制造他们自己的打印机，对设计进行改进，或通过定制满足他们特定的需求。这样一来，RepRap的设计得到了持续演进，打印质量和易用性都得到改善，而设备的成本也降低了。

RepRap项目也促成了若干与其相关的打印机的出现，它们与RepRap有一些共同之处，但放弃了自我复制的目标，优先考虑的是一些其他特性，如健壮性、组装难度等。这其中最知名的是MakerBot和

Ultimaker，它们使用了激光切割的胶合板框架。最新型号的MakerBot使用的则是用数控加工设备制作的钢结构框架。

6.5.2 软件

和激光切割的情况大体类似，对于该用什么软件包生成3D设计模型，我们没有明确推荐。如果已经对某个3D设计软件比较熟悉了，看看它是否能以打印机支持的格式输出文件。如果在使用打印服务，服务商会建议最好使用什么程序，或者告知它们能接受什么格式。如果这些办法行不通的话，也可以选择一个预算允许并且觉得最容易掌握的软件。

搞清楚怎样在一个二维的显示屏上设计三维物品不是件简单的事，因此最好认真学习一下所选软件的教程，这会让你在操纵对象方面得到最好的基础训练，确保构成设计的各组件能被正确排列，从而减小在打印物品中出现不该有的成分的风险。

Tinkercad (<http://tinkercad.com>) 和Autodesk公司的123D Design Online (<http://www.123dapp.com/design>) 是两个运行在浏览器中的设计工具。它们使你无需安装任何额外的软件就可开始做设计。

Autodesk公司还有一系列可供下载和安装的“123D”应用程序。你可以找到123D Design和123D Catch的桌面版本，后者是一个巧妙的应用程序，可以把某个物品的一组照片自动转换为该物品的3D模型，然后可能需要借助工具软件，如123D Design，对这个推导出来的3D模型做后续的改进。

SolidWorks (<http://www.solidworks.com>) 和Rhino (<http://www.rhino3d.com>) 都是工业标准的商用软件。SketchUp (<http://www.sketchup.com>) 在爱好者中比较流行，它曾在一段时间内为谷歌公司所拥有，但在2012年时被出售给了Trimble公司。

在开源阵营中，两个主要的竞争者是OpenSCAD (<http://www.openscad.org>) 和FreeCAD (<http://free-cad.sourceforge.net>)，前者采用了与众不同的脚本工作流。你还可以使用Blender (<http://www.blender.org>)，但它的学习曲线比较陡，更适用于制作3D动画，而不是计算机辅助设计。

设计完成之后，你需要用另外一个软件把它转换成一组将被送入打印机的指令。这通常被称为**切片算法**（slicing algorithm），因为该软件最重要的功能是把模型切割为一系列的层，并解决好如何指示打印机构建每一层的问题。大多数情况下，你所使用的特定切片软件，是由用来构建你的模型的特定打印机决定的，但对于诸如RepRap之类的开源打印机，你可能会有若干种切片软件可选。

Skeinforge是第一个被用于开源打印机的切片软件，但它在很大程度上已被较新、更易于使用的**Slic3r**取代了。这两个软件都允许你通过调整各种参数，对3D打印过程进行精细调整。可以指定的选项包括：塑料原料应该被加热到的温度、实心物体的填充密度、挤出机机头的移动速度，等等。

对于初学者而言，把这些参数都设定正确（或基本正确）是一项艰巨的任务。**Slic3r**提供了配置向导，能更好地指导你完成一个可用的初始设置。通过运行一些校准测试，可以对你用的那台打印机和正在使用的特定类型的塑料原料进行专门的设置。

当你很想把自己出色的设计打印出来时，先打印一个边长为20毫米的立方体可能会有点乏味无聊。但这样做能更容易地发现和纠正问题。花点时间让一切准备就绪是非常值得的，这将得到质量更好、更成功的打印作品。

6.6 数控铣削

数控（CNC）铣削和3D打印类似，但它的制造过程是做“减法”，而不是做“加法”。CNC是指铣头的运动由计算机控制，这和FDM 3D打印机的挤出机是由计算机控制的情况很相似。然而，数控铣的制造过程是对一块比成品大一些的原材料进行加工，把不需要的部分切削掉，而不是从无到有、一层层地构建出期望得到的模型。这就和雕刻家把一块石头上不需要的地方凿掉，从而让雕像显现出来一样。只是数控铣削用的工具是旋转的钻头（类似于电钻），而不是凿子。

由于切割原材料相对比较容易，数控铣床能使用的原材料的种类要比3D打印机多得多。虽然你仍然需要使用工业级的设备对硬化钢进行加工，但蜡、木头、塑料、铝，甚至低碳钢，都能很容易地用桌面式铣床加工。

数控铣床还能用来做更专业化（在制作电子装置原型时更有用）的工作，如制作定制的印制电路板（PCB）。你可以把你的PCB设计转换为一种适合用数控铣床加工的形式，而不是把它送到其他地方制作，或者用酸液蚀刻。也就是说，你可以用数控铣床把单板表面金属涂层不需要的部分铣掉，只留下导电通路。和蚀刻工艺相比，这种制作方法的一个优点是能同时完成任何元件孔和安装孔的钻孔工作，省掉了后续利用钻床手工钻孔的过程。

根据你需要的功能和预算，有众多类型的数控铣床可供选择。

从可以放在桌面上小型铣床，到工作台大小、以米计算的大型设备，数控铣床有一系列的尺寸规格。有的数控铣床甚至大到能占满一个飞机库，但它们往往是为完成某个特定任务而设计的，比如制作风力发电机叶片的模具。

但不是越大越好。随着铣床尺寸的增大，精确调整托板位置的难度也会增加，因此小型铣床通常能达到更高的加工精度。尽管不同数控铣床的加工精度有差别，但也只是高和极高之间的差别。数控铣床的加工精度经常能达到0.001mm的数量级，这要比当前的低端3D打印机高出好几个数量级。

除了尺寸和精度，不同类型的数控铣床之间的另一个主要的不同属性是能同时联动的轴的数量。

- **2.5轴**：虽然这种类型的铣床有X、Y和Z三个进给轴，但只能同时有两个轴联动。
- **3轴**：和2.5轴的情况类似，这种类型的铣床有一个能沿X轴和Y轴方向移动的工作台，和一个能沿Z轴方向移动的铣头，但它能做到三轴联动（如果加工指令要求它这么做）。
- **4轴**：这种类型的铣床是在3轴的基础上，增加了旋转轴，允许被加工工件围绕某个轴向旋转，通常是围绕X轴旋转（该回转方向被称为A轴）。分度轴允许把工件转动到一个指定的位置，以便对工件做进一步的铣削。例如，把工件翻转过来，对底面进行铣削。而完全可控的转轴允许在执行切削指令的过程中转动工件。
- **5轴**：这种类型的铣床增加了第二个旋转轴，通常是围绕Y轴旋转。该回转方向被称为B轴。
- **6轴**：增加了第三个旋转轴，使铣床具有了完整的改变位置的能力。如果是围绕Z轴旋转，这个轴就被称为C轴。

如同3D打印的情况，用于数控铣床的软件可以分为两类：

- **CAD**（计算机辅助设计）软件，用来设计模型；
- **CAM**（计算机辅助制造）软件，用来把模型转变为适当的刀具路径，即数控设备在加工工件时需要遵循的坐标列表。这将实现从原材料块到模型的转变。

刀具路径通常使用准标准化的数控编程语言G-code描述。尽管大多数的运动指令在不同的设备上通用的，但对于初始化设备之类的事情，不同设备在代码方面还是存在很大不同的。尽管如此，还是有若干第三方的CAM软件包可用。所以，运气好的话，你能从中选择一个使用。请参阅网址<http://lcamtuf.coredump.cx/gcnc/>，这里有对各种可能方案的扼要介绍，以及数控铣床入门知识相关的更多信息。

6.7 现有物品的循环再利用

到目前为止，我们只是讨论了如何完全从零开始创建一个新的物品。如果拥有装置所有构件的设计，并且了解它们是如何创建的，这将非常有益。但是，这些未必是所有原型设计方案最优先考虑的事情。

与构建物联网装置的其他组件相同，对于构件，从直接购买设计或实物，到完全由自己设计制作，你有诸多选择。正如人们一般不会考虑用铁矿石来自己制作螺母和螺栓一样，有时应该考虑重新利用比较复杂的机械装置或构件。

重新利用机械装置或构件的原因之一是顺便借用一下他人创造的规模效益。如果你需要的部件或全部配件有现成产品可用，买现成的通常要比自己制作更便宜。对原型制作来说，绝对是这样的情况。在产品生产阶段也可能是这样，不过那要取决于你要生产的产品数量。例如，**Bubblino**中使用的泡泡机就是一个儿童玩具上的现成组件。按照**Bubblino**当前的生产批量，相对于自己制造各种齿轮、风扇、泡沫环和外壳，买现成的泡泡机还是更便宜。

选择再利用的原因也可能是，你只打算制作几个成品，甚至只制作一个。在这种情况下，往往会需要搞懂电子单元的集成方式，以便和新制造的部件配合使用，或者需要了解如何对可重复使用的物品进行拆解，从而获得需要的部件。解决这些问题可能并不麻烦——因为你不会重复很多次。

对于单件制作的物品，常常会有意地把它们集成到现有的、大规模生产的产品中。**Russell Davies**委托艾德里安帮他制作两个具备最简单接口的WiFi音箱

（<http://russelldavies.typepad.com/planning/2011/07/secondary-attention-and-the-background-noise.html>），并且要的是两个不同的装置。一台音箱需要从头开始做，使用全新的电子单元和激光切割的外壳，而另一个则是经过改造的1974年制造的晶体管收音机。收音机的电路已被移除，以腾出空间放置小尺寸的ARM Linux单板，原来的扩音器被保留了下来，波段选择开关也被重新配置用作程序的界面。当所有部件

都安装到“收音机”中之后，我们看到了一个非常熟悉的物品，但它却已经拥有了全新的功能。

原型设计是探索和完善你的想法的一个过程，我们之前介绍过这个理念。对现有的物品进行重用和再利用，其最大的用途可能就是践行这一理念了。

考虑到在原型设计阶段，需要对各种想法进行快速迭代，所以，任何有助于缩短制作周期，能让你更快验证自己理论的措施都是有用的。例如，在考虑一个有连网能力的床头柜的用户交互方式时，把一块 **Arduino** 单板绑定到闹钟上，就能提供一个足够好的近似物，让你尝试不同的方案。

如果最终设计所需要的工艺流程预计会花费大量的成本（如制作注塑模具），或者该设计要求设计师具备较高技能，而暂时又没有财力马上雇用一名这样的设计师，也许可以把一个已经存在的、功能相似的产品用作替代物。这样，该项目就能继续推进，从而达到一定的阶段，让人们觉得值得继续进行投入。

当然，能重用的不仅仅是某个现成的实物。**Thingiverse** 网站（<http://www.thingiverse.com>）是一个各种设计的资源库，其中的大多数设计是3D打印或激光切割相关的，并且这些设计根据知识共享（**creative commons**）协议，都是可获得的，允许你按照原样使用这些设计，或对这些设计进行修改或扩展，以适用于你自己的需求。

案例研究：阿克斯钟

艾德里安的**MCQN**有限公司最近完成了一个项目，该项目把本章探讨的几个主题结合到了一起，了解一下这个项目对我们来说可能是有益的。

阿克斯钟是一个能连接到因特网的钟，是大数据创业公司 **ScraperWiki**（<http://scraperwiki.com>）委托**MCQN**制作的。这个装置与该公司的在线账单系统相连。每当有一笔新的支付款进入公司的银行账户，阿克斯钟就会响。这样就能进一步地激励销售团队实现更多的销售，同时让每个人都有机会为每一次的成功庆祝。

以上就是对这个项目的简介。现在让我们看一下它的设计和实现手段。从头开始铸造一口钟通常不太可行，所以第一步就是调查一下有什么样的钟可以从其他地方获得，并被重用。

与铸钟方面的专家朋友进行初步讨论后，他们很快发现了一些很不错的音调已经调整好的钟，但遗憾的是这些钟的价格超出了可用的预算。之后他们把目光投向了拳击比赛场地用的铃铛，还考虑对老式的电话铃进行再利用。这两种类型的铃铛之所以具有吸引力，是因为它们都同时包括了机械装置和电气敲击装置。然而，在与客户沟通之后，他们最终选定的是一口传统的船上用的铜钟。由于ScraperWiki和艾德里安的公司都在利物浦这个港口城市的缘故，这口铜钟更能被ScraperWiki所认同。

得到铜钟之后，下一步就是解决怎样安装它的问题。鉴于MCQN的丰富经验，电子单元的开发相对比较容易。他们很快就实现了这个功能：当在线事件发生时，Arduino Ethernet板就会触发与之相连的电磁阀动作。他们只需要想出一个办法，把电子单元和铜钟这两个部分装配到一起。

图6-7这张照片，显示的是艾德里安在他的Moleskine笔记本中画的一些初始的草图。通过这些草图，艾德里安对构造外壳的可能方式和一种可能的接头设计的细节做了一番探索。

他们认为木材是和钟的黄铜材质实现互补的最佳选择。不同于普通的在激光切割机使用的桦木胶合板，他们想要的是深色的木材。此外，可能的接头设计方案要求木材的厚度要远大于桦木胶合板3mm的厚度，因此他们最终从本地木材商提供的一系列的硬木材料中，选择了8mm厚的橡木。

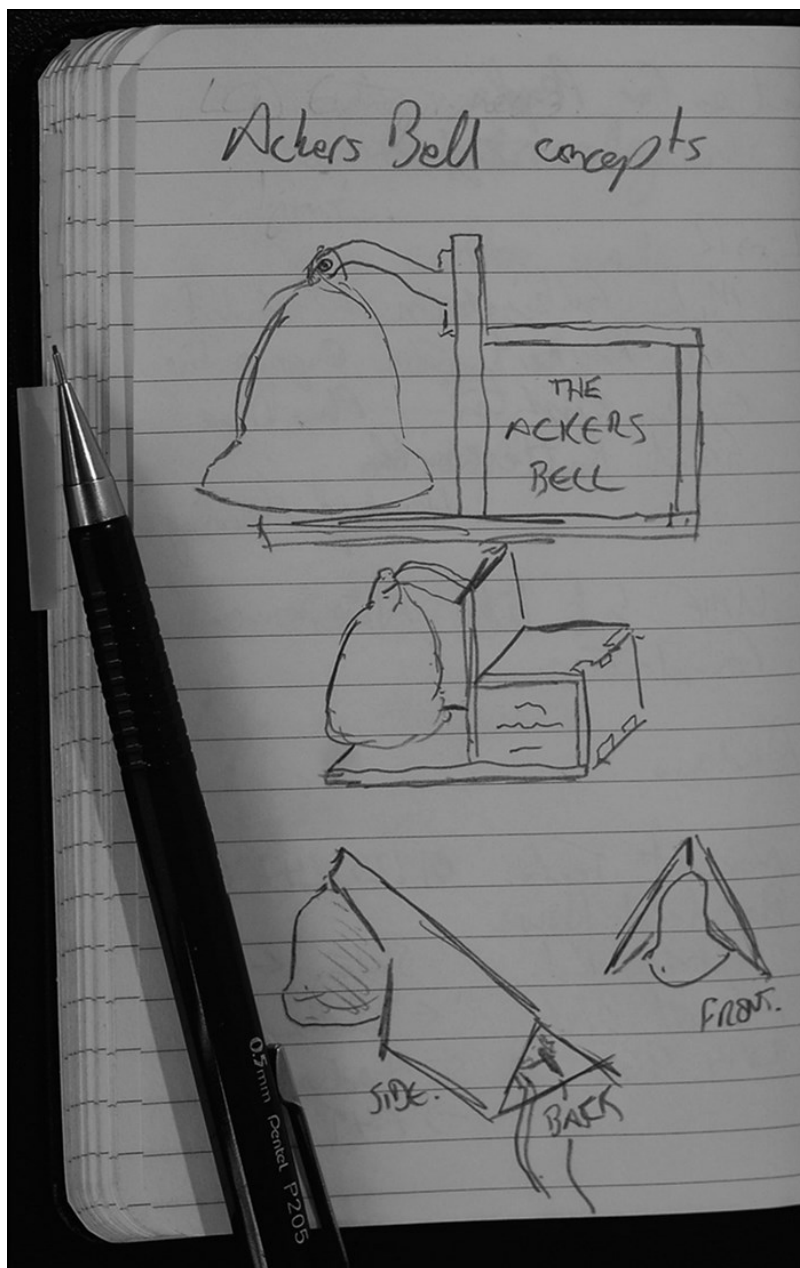


图 6-7 阿克斯钟早期的设计想法

遗憾的是，几次切割测试（或者更准确的说是灼烧测试）表明，橡木太硬了，现有的几台激光切割机都无法完成切割。因此还需要继续尝试其他材料。

在木材的选择上，他们在桦木胶合板表面尝试使用了各种不同的饰面，也尝试使用了一些着色剂和蜡。最终发现，在桦木胶合板

上涂覆黑色的蜂蜡，不仅能产生好的色泽，还能带来额外的好处，能为木板提供一些保护和处理。

至于如何使用这些木板的问题，在公司实习的利物浦大学产品设计专业的学生伊恩·斯科特花了一些时间将其解决。他的设想是把钟放置在一个栅格状的框架中，该框架使用普通的3mm厚的桦木胶合板制成，其形状和钟本身的轮廓线相仿。这意味着，尽管框架最终的外形被设计为相当复杂的三维结构，但它是可以用激光切割机切割的木片构造出来的。

除了提供用于悬挂铜钟的支撑物，这个采用激光切割的设计还包括一个用于放置Arduino板的平台和一个电磁阀的安装点。这些构件都被巧妙地隐藏到了铜钟的内部，从而保持了外观的整洁。然而，艾德里安所选用的那个电磁阀在断电时，没有什么能阻止敲击杆继续移动的机构，因此，敲击一次之后，弹簧的复位会导致敲击杆脱落。

虽然他们可以毫无疑问地用激光切割出某种类型的挡板，但为了实现一个更为整洁的解决方案，他们决定使用3D打印机制造这个部件。伊恩设计了一个相当简单的L型的部件，它可以和电磁阀使用同一个安装点，能提供一个由黑色的ABS塑料制成的屏障，从而使敲击杆的运动范围受到限制。

设计工作的最后一步是确定电磁阀的位置，使得它能在敲击铜钟时产生清晰、连续的钟声。中看不中听的钟是完全没用的。

他们之前对此做了测试，并且在安装点留出了一些调整空间。但当所有部件都被正确的组装好后，他们才发现之前做的测试是不够充分的。虽然铜钟有时能产生清脆悦耳的共鸣音调，但时常出现的情况是，敲击杆要么很难击打到铜钟，要么击打的力度太大，以至于铜钟会在敲击后回荡回来碰到电磁阀，使得声音受到抑制。

在最初设计框架的时候，如果他们能对安装电磁阀的方式做更多次的迭代，这种问题就更容易纠正。想在构造过程的后期解决这个问题，能做的改变其实非常有限。这意味着他们要耗费更多时

间，用来做各种尝试，例如，增加电磁阀的输入功率，或者在保持现有构件位置不变的情况下，设法对安装点做些改动。

最终，他们制作出一个垫片，用来对电磁阀敲击钟的角度进行调整，使得敲击杆在与钟接触时能保持与钟的表面垂直。你不得不相信我们的这个说法，即阿克斯钟现在的声音很好听。虽然你通过本书无法听到，不过，我们至少能让你见识一下它漂亮的外观，见图6-8。



图 6-8 准备交付客户的阿克斯钟

6.8 小结

本章（和之前的第5章）为你构建自己的物联网装置打下了良好的基础。

对设计做一些迭代和探索，能让你对设计的外形和尺寸如何影响其在人们生活中的使用方式有更好的理解。堆满你工作室的一系列的原型能很好的印证你的收获。

除了展示设计思想的发展过程，从用泡沫板或纸板做的实体模型，到使用激光切割机或3D打印机等快速原型制作工具制作的更坚固、更精致的模型，各种原型也呈现了一个逐步改进的过程。

对于你的物联网产品，在介绍完原型系统的装置部分后，剩下的就是一些与之进行通信的在线服务了。下一章将引领你了解这些内容，看一下怎样与现有的在线服务进行通信以及怎样开发全新的在线服务。

第7章 原型系统在线组件的设计

第2章已经介绍了物联网装置是如何成为魔力物品的：物理装置中安装的嵌入式控制器和传感器能使该装置做一些具有智能的事情，而这些事情是纯粹的实体或机械物做不了的。即使只具备控制器和传感器，你就已经可以看到一些有魔力的物品了，例如，空气清新机只在检测到有人走过时才会向屋内输送香气。但物联网这个术语的确表明，因特网也是它的组成部分。

显而易见，每个组件都扮演了重要的角色。设计出来的物联网装置体现了与环境、可见性等有关的设计原则（在第2章已介绍）；控制器和相关联的电子单元实现了对真实世界的感知，并且能对外界施加影响；因特网增加了一个通信的维度。网络组件使得你或其他人能从物联网装置处获得事件通知，或者能让你实时地依据装置收集的数据做出适当应对。它允许你对来自于不同位置、不同类型的传感器信息进行聚合。同样，网络组件扩展了你的影响范围，这样你就可以从远端控制或激活装置，它也使得网络世界可以用新鲜有趣的方式渗透到物理世界中。

因此，记录温度的传感器装置可以把数据写到Xively网站，Bubblino这样的通知装置能以吹泡泡的形式响应推特上推文的发布。尽管我们在第2章介绍过一项基本的设计原则，即物联网装置应该是因特网上的一等公民，但实际上，它们目前看似是要与某个特定的网站或服务紧密绑定的。这样做的理由是：不同于计算机、平板电脑或手机等通用装置，物联网装置是为了实现某个目的而设计的，并不一定拥有能便于修改配置的键盘和屏幕。

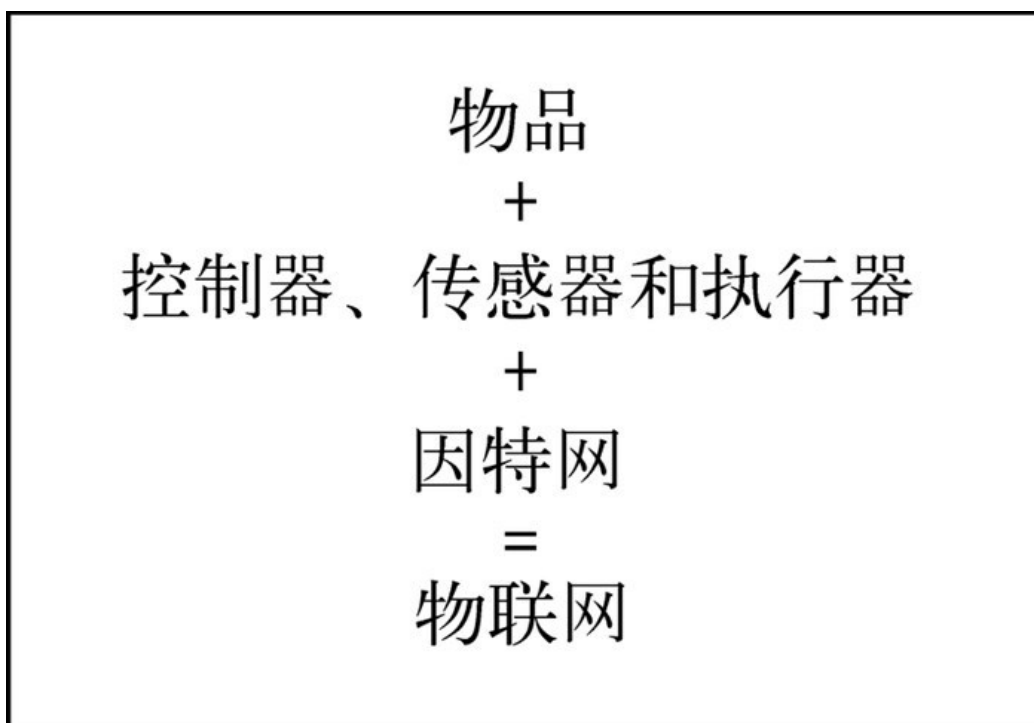


图 7-1 物联网的关键组件

在不久的将来，物联网装置之间，物联网装置与其他局域网或个人区域网上的应用程序和计算机之间，将很可能会使用标准化的协议通信。但在目前，在我们介绍的大多数例子中，各个物联网装置被连接到了一个单独的Web服务。尽管你已经对现有的服务（Xively、Twitter）有所了解，自己创建一个服务还是会让你从中受益。对于个人项目，是否创建这样的服务可能并不重要。但如果是在开发一个准备出售的产品，可能还是会希望能对服务有所控制。否则，一旦所使用的网络服务停止运作，或者改变了服务条款，就会让你受到责难；或者，改变了编程接口，就会使得代码停止工作，可能就必须召回所有装置并重新对它们进行编程。实际上，即便是Bubblino的网络访问，也是经由位于<http://bubblino.com>的一个网络服务。该服务使得用户可以根据个人需要进行设置，使Bubblino能搜索特定的词。同时，这个网络服务给了艾德里安更多的灵活性，如果情况发生改变， he 可以把所有的Bubblino重定向到一个不同的服务或应用编程接口。

7.1 开始使用API

对于物联网装置而言，Web服务最重要的部分是应用编程接口（API）。API是面向机器而不是人提供的获取服务的方式。回想一下获取因特网服务的经历，你可能要遵循若干步骤。例如，为了查看Flickr上某个朋友的照片，你可能要做以下的事情：

- (1) 启动Chrome、Safari或IE浏览器；
- (2) 用Google搜索Flickr网站，并点击相应的链接；
- (3) 输入用户名和密码，点击“登录”；
- (4) 浏览网页，并点击“联系人”链接；
- (5) 从联系人列表页面开始，再点击几次链接，直到找到要找的人所在的页面；
- (6) 向下滚动页面，找到所需要的照片，然后点击这张照片。

尽管对人而言，这些动作很容易完成，但其中涉及了很多的查看、思考、键入和点击行为。计算机不能像人一样进行查看和思考。一旦Flickr对用户接口稍作改动，上述复杂、冗长的，需要遵循一系列操作并对每个页面做出响应的过程很可能就会失效。例如，如果Flickr把“登录”（Login）改写为“登入”（Sign in），或者把“联系人”修改为“朋友”，一般人甚至很可能注意不到这些改动，但一个典型的计算机程序却会因此完全失效。相反，计算机能非常愉快地调用预定义的各种命令，如login或get picture #142857。

7.1.1 API的混聚

也许你想要的数据已经可以从因特网上获取了，但其呈现形式不符合需求。那么你可以通过混聚（mashing up）多个API以获得结果，这能用来创造很好的效果。举例如下。

- 使用地图API绘制租售物业的位置。例如，谷歌地图经由Craigslist显示待出租的物业位置；或者，像伦敦Foxtons公司一样，使用Mapumental显示其物业的位置。
- 使用时间线或图表API，在世界地图上显示推特趋势话题。
- 利用《卫报》的API提取新闻头条，并从Flickr网站获取相关的图片。

你需要完整的API吗

对于个人项目而言，最好能在开始时利用现有的服务，如前面提到过的Twitter和Xively，伦敦交通局提供的租赁自行车使用情况的API，或mapme.at。

正如Bubblino的情况，也许你将来会对服务做些扩展，增加简单的配置功能，封装自己的API。但如果你想与之交互的数据尚不存在，这可能是一个创建有通用价值的新服务的机会。

一些关注度较高、易用性较好的API（如谷歌地图API）希望把数据嵌入其产品当中。这意味着它们最适合在Web浏览器中使用，但由于最终产品不受你的控制，所以从微控制器访问这些API可能会受到一定限制。

7.1.2 Web数据抓取

在很多情况下，一些公司或机构能获得很好的数据，但并不想让别人通过API获取，或者不具备实现此类API的资源或知识。你从上述Flickr的例子中可以看到，让计算机冒充浏览网站的用户，通过查找用户界面元素的方式定位到正确的位置，这种实现是容易失效的。但这并不意味着这样做是不可能的。我们通常把这种方法称为屏幕抓取（screen-scraping），这个称谓可能多少有点贬损的意味。下面是几个实际的例子。

- 艾德里安通过屏幕抓取的方式，从船舶自动识别系统网站（www.shipais.com/，该网站的数据是航运爱好者们采用半手工的方式标绘的）获取了莫西河上的船舶数据，然后把这些信息用

@merseyshipping账号

(http://www.mcqn.com/weblog/connecting_river_mersey_twitter) 发布到了推特上。在谈及这个项目时，他将其称作“把河流连接到因特网”的一种方式。因此，尽管该项目本身不存在一个连接到因特网的装置，但还是可以认为它是属于物联网的范畴。

- 公众监督网站 (<http://www.publicwhip.org.uk/>) 就是通过使用Web数据抓取工具读取英国政府会议的议事录（以Word文档的形式发布）来实现的。利用所得到的数据，该网站能生成人和机器都可读的提要信息，让公众知道他们选出的代表是如何投票的。
- 除了其他的用于处理在线数据的工具，ScraperWiki网站 (<https://scraperwiki.com>) 有一个优秀的用于编写数据抓取工具的平台。该平台支持若干种动态编程语言，能抓取数据并将其放入数据库表中。实际上，该平台提供了“机械化”生成数据抓取脚本的基础设施，从而能把那些枯燥、重复性的制作数据抓取脚本的工作外包给ScraperWiki，而这些脚本可以在你自己的计算机或服务器上运行。ScraperWiki的首席执行官Francis Irving在自己的一个物联网项目中用到了这个平台。通过抓取利物浦市政服务机构网站上相关页面的数据，他查找到了垃圾收取时间。他的Binduino装置 (<https://github.com/frabcus/binduino>)，即一个基于Arduino定制的电子装置，能定期检查这些抓取的信息。当收垃圾的时间到了，Binduino会点亮一些电致发光线，让垃圾桶发光。

7.1.3 合法性

屏幕抓取可能会违反网站的服务条款。例如，谷歌不允许抓取其搜索页面，但提供了相关的API。即使不考虑法律的制裁，但如果违反了像谷歌这样的公司的服务条款，他们可能会拒绝你使用该公司的其他服务，这将会带来很多不便。

另外一些数据受版权（或数据库权限等）的保护。这里要讨论的一个项目是一种数据抓取工具。该工具能读取足球比赛的赛程，并且当你的球队正在比赛时，该工具能让一个罗盘的指针指向比赛地点的相对方位。但在英国，赛程表显然是有版权的。英格兰和苏格兰的足球协会已经对各种不向他们支付许可费的经营机构提起诉讼

(<http://www.out-law.com/page-10985>)。对于个人的小项目，创建一个这样的数据抓取工具不算什么大问题，但这样做可能会减小该项目成为商业产品的可行性（这取决于许可费能否算作合理的商业成本，不要高得无法承受）。

另外，也可以经常利用一些替代的信息源。例如，你可以用 **OpenStreetMap** 替代谷歌地图。英国的邮政编码数据库是受皇家版权保护的，但它也有其他一些可能不太完整的众包版本。

有关数据合法性方面的进一步讨论，以及一些其他事项，请见第9章。

7.2 编写新的API

假设所要使用的数据无法获取，或者不能利用其他现有工具和数据源轻易实现混聚或抓取，那么也许应考虑创建一个全新的信息源或服务。比如说，可以从免费资料或现有的授权资料中收集数据，并对其进行处理。另外，现有的物联网装置也许就能够生成此类数据！

我们将使用示例项目**Clockodillo**，带你了解构建自己的API的过程。这是哈基姆构建的一个物联网装置，用来帮助他使用番茄钟工作法¹（www.pomodorotechnique.com/）。

¹ 介绍番茄工作法的时间管理书《番茄工作法图解》已由人民邮电出版社出版。——编者注

通过番茄工作法，可以把工作任务分割为25分钟的时间块，并且使用厨房定时器来追踪你完成任务花费的时间，避免在每个25分钟的时间块内分神。

Clockodillo探索了如何借助物联网达成这一目的。它把厨房定时器连接到了因特网，使得对任务的跟踪更加容易。与此同时，**Clockodillo**保留了实体定时器的简单性（能通过扭转设定时间并启动定时），并且在计时过程中显示剩余多少时间。

到本章结束时，将实现一个能与定时设备相连的实际API的大体框架。

尽管设计在浏览器上使用的Web应用程序时，会将用户将要完成的操作和浏览应用程序的流程结合在一起，但在编写后端API时，却需要更多思考如何处理数据的问题。

正如享有盛名的软件工程师弗雷德里克·布鲁克斯所言：

给我看程序的流程图而藏起数据表，我仍将莫名其妙。但给我看数据表，我就不再需要流程图了，因为流程将显而易见。

——《人月神话》

当你知道自己拥有什么数据，能对这些数据做哪些操作，以及这些操作将会返回什么数据之后，应用程序的流程就变得简单了。这是思考编程的好机会，你无需从一开始就去担心用户界面或交互的问题。尽管这可能听上去与编写Web应用有很大不同，但它实际上是一种理想的开始方式：通过从前端分离出业务逻辑，可以去除模型（核心数据结构）与视图（HTML/JavaScript）和控制器（窗口小部件、窗体交互，等等）之间的耦合状态。如果你用过某个流行的“模型-视图-控制器”（MVC）框架（Ruby on Rails、Django、Catalyst等）编写程序的话，那么就能了解这种方式的优点了。

如果用这种方式开始API设计，以后就可以轻松地添加网站。这是这样做的最大好处。这点将在7.2.5中进一步得到展示。

7.2.1 Clockodillo

正如之前介绍的那样，Clockodillo是一个连接到因特网的任务定时器。用户可以通过转动转盘，设置一定的分钟数，然后定时器开始工作，直到定时结束。该装置还能向API服务器发送消息，从而使服务器了解一个任务的状态：已开始、已完成，还是已取消。

对于一些API交互，它们做的事情与物理装置的功能相同：

- 启动新的定时器；
- 改变现有定时器的定时时长；
- 标记定时器已完成定时；
- 取消定时器。

某些与定时器数据结构的交互，例如任何需要显示器或键盘的交互，对于主要由转盘构成的装置而言太过复杂，以至于无法在装置上显示。那种交互可以通过计算机或手机上的应用程序完成。

- 查看和编辑定时器的名称/描述

自然，用户可能希望能查看历史数据：

- 之前使用过的定时器列表
 - 它们的名称/描述
 - 总的定时时长和它们是否被取消

假设计划构建的不仅仅是一个装置，那么还需要某种识别装置的机制。在第10章介绍扩大规模转入实际生产阶段时，我们还会回来谈谈这个有趣的话题。现在只需假设每个装置会发送某种识别标记，如MAC地址——正如你在第3章所见，MAC地址是每个网卡芯片都具备的独一无二的编码。

这样用户就能设法让服务器识别自己。这样做了之后，所有上述操作将只与一个给定的用户ID相关联。

7.2.2 安全

我们是否遗漏了某些东西？假如你大喊道：“安全！安全性哪儿去了？”那说明你考虑得非常周全。在很大程度上，安全的重要性取决于被传递信息的敏感程度，以及信息泄露是否符合利益。就Clockodillo而言，也许某位老板想复查一下雇员是否在使用这个定时装置；或者，某个竞争对手可能想通过查看定时任务的描述信息来窥探你的公司的运作情况；又或者，某个（名声不太好的）竞争对手企图通过输入假数据的方式对服务进行干扰破坏，意图毁坏你公司的声誉。

如果服务涉及健康或财务信息，对破坏者来说，它可能会成为一个更具吸引力的目标。个人的位置信息也很敏感：知道你何时外出，对窃贼而言可能是颇具实用价值的信息。

安全是一个非常重要的问题，设计API时需要牢记这一点。但在设计的开始阶段，需要先弄明白你到底想用这个API做什么。

任务	输入	输出
(1) 创建新的定时任务	用户标识、定时时长	定时器编号

任务	输入	输出
(2) 改变定时任务的时长	用户标识、定时器编号、新的时长	OK
(3) 标记定时器为完成状态	用户标识、定时器编号	OK
(4) 取消定时任务	用户标识、定时器编号	OK
(5) 为定时任务添加描述信息	用户标识、定时器编号、描述信息	OK
(6) 获取定时器列表	用户标识	定时器编号的列表
(7) 获取定时器的信息	用户标识、定时器编号	描述信息、创建时间、状态

显然，这些API请求必须传递用以识别用户的具体信息，这就是身份识别的问题。也就是说，应用程序需要知道是为哪个用户创建的计时器，以便用户以后可以重新找回相关的信息。

而应用程序也应该对这些请求进行身份验证。对于不是特别敏感的信息，用密码做身份验证就“足够好”了。

观察上表，你可以看到，任务(4)对应的请求可以由物理定时器装置发出。传递描述信息，显示定时器列表的具体信息，或者获取与定时器相关的信息，需要具备比定时器装置更多的输入输出能力。

但对于任务(4)，怎样才能让定时器装置传递用户名和密码呢？用户可以经由USB接口，用计算机配置这些信息。但这样做可能会比较复杂，意味着该装置需要具备某种持久化的存储机制。这也表明有更多工作要做，需要有更强大的微控制器，对于低端控制器，可能需要一个额外的SD卡读卡器。

一种常用于微控制器的技术能让微控制器发送一个物理ID，这通常是指它们的MAC地址（分配给每个联网装置，或更确切地说是分配给它们的每个网络接口的唯一ID）。因为这个ID是唯一的，所以可以把它与某一用户绑定。

此外，不管发送的是MAC地址还是用户名和密码，你必须考虑在因特网上发送身份识别或身份验证数据的风险。回想一下第3章中有关因特网路由的介绍，你已经知道，携带请求消息的数据分组可以经由各种路径传送。如果用户名和密码采用了“明文”形式，则任何实施数据包嗅探的人都可以读取它们。主要情境有以下两种。

- **攻击者是针对某个特定的用户，并且能接入该用户所在的有线或（未加密的）无线网络**。攻击者能读取和使用详细信息（创建虚假的定时器或获取有关该用户的信息）。
- **攻击者能接入某一个中间节点**。攻击者不针对某个特定的装置，但可能在观察有什么未经加密的数据通过，有没有诱人的攻击目标。

当然，你的物联网装置可能尚未成为一个诱人的目标。我们希望如此，然而这种状况可能是暂时的。如果你的装置变得很受欢迎，就会有竞争对手乐意出手，暴露其安全漏洞。虽然软件产品可以轻易地通过升级来应对这种情况，但升级硬件项目需要处理的事情则要复杂得多。

更糟的是，如果软件密码被泄露了，网站可以方便地提供修改密码的途径。虽然这项工作对于具备显示器和键盘的计算机而言很简单，但对物联网装置来说情况就不一样了。因此你需要一种方法来配置设备，以更改它的密码，例如，使用一个托管在服务器上或装置自身中的Web控制面板。这个解决方案比较麻烦，需要装置具备本地存储单元，以便写入新的密码。

对于用明文发送密码这个问题，一个显而易见的解决方案是对包括身份验证信息在内的整个请求消息加密。对于一个Web API，通过指向https://而非http://开头的网络地址，你可以简单地实现这一目的。你不需要对应用程序代码做任何进一步的改变。对于大多数Web服

务器，对其进行设置，使其支持HTTPS的过程都非常简单。我们将在本章的示例代码中进一步介绍。

在装置上解决这个问题可能会比较困难。加密运算需要求解非常大的方程组，并且要占用CPU和内存。例如，当前的Arduino平台就不具备HTTPS库。尽管功能更加强大的微控制器支持HTTPS（而且毫无疑问，将来的Arduino版本也会支持），但未来的微控制器还是会存在类似的限制，这点你能很容易地想象到。

如果物联网装置可能会被攻击者恶意利用，那么在选择平台时，安全通信就是一个关键的度量因素。对于某些数据，例如空气质量监测仪收集的数据，攻击者不大可能从中获取有用的信息。但有些数据可就敏感多了，假如可以从一些数据中推断出家中是否有人，或可以利用某些数据来控制物品或进行其他操作，那么你就需要确保这些数据的安全了。

如果你在定义自己的API，可以考虑采用Arduino的加密库，用加密库实现自定义的安全通信是可行的。如果你决定采用这种方法，就需要在安全专家的帮助下谨慎地处理。

Twitter等网络服务使用了OAuth 1.0协议，该协议允许第三方应用程序无需密码就能访问你的账户，是在不使用HTTPS的情况下提供强身份验证的范例。API调用的内容仍然采用明文方式传送，因此实施网络嗅探的攻击者能看到正在传送的消息，但他不能对请求消息进行修改或重放。为了给OAuth 1.0增添加密机制，防止其他人窥探正在传送的消息，你仍然需要在HTTPS之上运行OAuth 1.0。OAuth 1.0使用指南对它所解决的问题进行了有用的讨论，请参见

<http://hueniverse.com/oauth/guide/security/>。

为了避免使API的讨论变得复杂化，作为一种折衷方案，在本例中，我们建议对于任何包含用户名/密码的Web请求，都要坚持采用HTTPS。但对于由定时装置发送的前4项请求，则允许采用HTTP发送MAC地址。

修改后的请求列表如下所示。我们添加了2个请求消息，用来添加和核对某个用户的MAC地址，并根据之前定义的请求所适用的资源类型，对它们进行了分类。

任务	身份验证方式	输入	输出
(1) 创建一个新的定时任务	MAC地址或用户名/密码	定时时长	定时器编号
(2) 改变定时任务的时长	MAC地址或用户名/密码	定时器编号, 新的时长	OK
(3) 标记定时器为完成状态	MAC地址或用户名/密码	定时器编号	OK
(4) 取消定时任务	MAC地址或用户名/密码	定时器编号	OK
(5) 为定时任务添加描述信息	用户名/密码	定时器编号, 描述信息	OK
(6) 获取定时器列表	用户名/密码		定时器编号的列表
(7) 获取定时器的信息	用户名/密码	定时器编号	定时器的信息
(8) 用户注册一个装置	用户名/密码	MAC地址	OK
(9) 获取用户装置的细节信息	用户名/密码		MAC地址

如你所见，前4项任务可以在定时装置上设置。但与定时装置相比，在客户端（网站或本地）能实现更多的方法。通常的做法是：装置负责提供若干其输入输出模块特别适合实现的功能，而一整套其他功能，包括对数据的支持，对身份验证的控制，以及对数据的呈现和编辑，则可能需要由功能更为丰富的输入装置（可能是另一个智能装置，或是计算机及手机等通用计算装置）来提供。

7.2.3 API的实现

API定义了从客户端到服务器端，以及从服务器端到客户端所传送的消息。最终，你能以任何你想使用的格式发送数据，但一般来说，使用某个现成标准应该是更好的选择，因为在客户端和服务端，对于生成和理解所需要的消息，都有便捷的库存在。

几个最常见的值得你考虑的标准如下。

- **表征状态转移 (REST)**：使用HTTP方法，包括GET、POST、PUT 和DELETE，访问一系列Web URL，例如，
`http://timer.roomofthings.com/timers/`或
`http://timer.roomofthings.com/timers/1234`。得到的结果通常采用XML或JSON格式，但实际的格式常常取决于HTTP的内容类型协商机制。
- **JSON-RPC**：访问单一Web URL，如
`http://timer.roomofthings.com/api/`，传递一个JSON字符串，如
`{'method':'update','params':[{'timer-id': 1234, 'description':'Writing API chapter for book'}], 'id':12}`。其返回值也是JSON格式，如
`{'result':'OK', 'error':null, 'id':12}`。
- **XML-RPC**：该标准与JSON-RPC类似，但用XML代替了JSON。
- **简单对象访问协议 (SOAP)**：与XML-RPC类似，该标准的传输格式是XML。但它提供了额外的功能性的层级，对于非常复杂的系统可能有用。

Jason和远程过程调用

我们将在这里对上面提到的几个缩略语做一个简短的介绍。它们并不妨碍你对上下文的理解，但对其进行简要说明无疑是有益的。

- **JavaScript对象表示法 (JSON)**，其发音与“Jason”相同，是格式化数据的一种表示方式，用来在不同系统间比较简便地交

换数据。正如其名称所暗示的，JSON起源于JavaScript编程语言，但现如今在Ruby和Python等其他语言中也能很容易地使用JSON。

JSON的核心部分由一系列的属性构成，其语法形式如下。

属性名：属性值

属性值可以是字符串、数字、布尔值（**true** 或**false**）、另一个JSON对象或数组（一系列对象）。

各个属性之间用逗号分隔。一组不同的属性可以用{} 组合在一起，构成一个对象。一系列相同类型的对象可以用[] 组合在一起，构成一个数组。

例如，对于一个包含2个对象的数组——每个对象包含一个名字属性和一个年龄属性——它看上去应该是这样的：

```
[
  { "name": "Object 1", "age": 34 },
  { "name": "Second object", "age": 45 }
]
```

想了解全部的细节，请访问JSON的官网，<http://json.org/>。

- 远程过程调用（RPC）这个术语是用来描述调用其他计算机（不是你编写代码所在的本地计算机）上程序代码的方式。我们一直讨论的Web API是RPC的一种形式。由于Web这个词能更好地解释远程通信的实现方式，所以在这种情况下往往不用提及RPC这个术语。
- 可扩展标记语言（XML）。就本书讨论目的而言，可以认为它是JSON的一个替代物。它使用< 和> 为各个元素划界，其语法往往比等效的JSON更加冗长。因此，它不太适用于资源受限的系统，如物联网装置。我们推荐你使用JSON。XML和

HTML有共同的起源，因此，如果熟悉HTML，那么肯定不会对XML陌生。XML是由万维网联盟（W3C）制定的，HTML、CSS和其他Web标准也都是由该组织负责的。可以去该组织的官网上了解有关XML的基础知识（www.w3.org/standards/xml/），进而能够学习更多相关内容。

我们推荐你使用REST，但完全有理由使用另外的标准。例如，如果试图复制另一个XML-RPC服务的接口，或者因为在从事其他项目时，已经对SOAP非常熟悉了，那么这些理由将胜过其他考虑因素。

我们在本章中将使用REST API，因为它广受欢迎，有良好支持，并且对于资源受限的微控制器来说，其交互过程也比较简单。然而，在我们所介绍的设计考量中，大部分都适用于其他标准。

REST也存在一些不利因素。例如，对于一些HTTP方法，不是每个客户端或服务器都能良好地支持它们。特别是Web浏览器仅对GET和POST方法有原生支持，这使得在网页上使用REST进行交互会将事情复杂化。

在REST的最佳实践方面也有许多分歧。REST专家们未必总是支持最实用的解决方案。本书不是有关REST的图书，而旨在提供一种使用方法。我们将指出那些出于权宜之计做决定的地方。

在REST中，每个资源都被附加到了一个URL上，你操作的是URL。例如，与编号为1234的定时器交互时，你可能要访问/timers/1234。创建一个全新的定时器，你将会访问/timers。如你所见，你会用到不同的方法，这取决于你是想获取一个资源（GET），还是想先把一个资源递交到服务器上（POST），还是想更新某个资源（PUT），或者想从服务器上删除某个资源（DELETE）。

授权和会话管理

上表中的内容会让人们认为，每次发送请求时都要传递用户名和密码，但这不是一个真正的好办法。如果攻击者成功破解了事务处理流程，那么用户名和密码都会被其获得。更好的办法是只执行单次登录，然后在后续请求中传递某种形式的令牌。这种方法

可以在限定时间之内使用，或者限定在一次会话中使用。就REST而言，我们试图尽可能全面地使用HTTP的功能。这样做，会话cookie会被选定为令牌。大多数服务器和客户端程序能自动处理cookie，因此在后续请求中只需检查cookie。尽管这听上去很好，但在当前，Arduino的HttpClient库还不支持cookie。毫无疑问，这个问题将会很快得到解决，或者能找到某种变通的方法（通过手动解析和设置HTTP报头）。但对于这个定时装置的例子，你可以继续在发送每一个请求时传递MAC地址。

REST API最终将如下表所示。

资源URL	方法	身份验证方式	参数	输出
(1) /timers	POST	MAC地址或Cookie	定时时长	定时器编号
(2) /timers/:id/duration	PUT	MAC地址或Cookie	定时时长	OK
(3) /timers/:id/complete	PUT	MAC地址或Cookie		OK
(4) /timers/:id	DELETE	MAC地址或Cookie		OK
(5) /timers/:id/description	PUT	Cookie	描述信息	OK
(6) /timers	GET	Cookie		定时器编号的列表
(7) /timers/:id	GET	Cookie		定时器的信息

资源URL	方法	身份验证方式	参数	输出
(8) /user/device	PUT	Cookie	MAC地址	OK
(9) /user/device	GET	Cookie		MAC地址
(10) /login	POST	用户名/密码	用户名/密码	Cookie + OK
(11) /user	POST		用户名/密码	Cookie + OK

所有这些前期工作，对于在头脑中厘清物联网装置和服务的交互过程是至关重要的。尽管作为示例，我们的确给出了一个粗略的不完整的实现，但实际上，编程实现这一交互过程超出了本书的范围。在编程实现方面，不存在单一的最佳解决方案，并且很多选择取决于你的编程专长。或者，如果你是雇用一名开发者做后端的开发工作，这些选择将取决于你是否能找到合适的人。

在选择Web后端平台时，应该考虑如下一些因素：

- 你已经熟悉什么平台（如果你打算自己做开发）？
- 本地或因特网上的招聘市场是什么行情（如果你打算把开发外包出去）？
- 某种编程语言的发展势头是否良好？它的开发活动是否活跃？它是否有一个健康的社区（或商业支持）？是否存在良好的生态系统，有丰富的程序库可用吗？

我们故意没有提及“能力”或“速度”等因素。因为几乎任何的、能实现上述最重要的Web标准的编程语言都足够强大，足以实现我们需要的Web应用。编写Web应用时，你最关心的可能不是能力或速度，而是开发速度、健壮性和可维护性。当然，如果你的应用在规模上需要做足够大

的扩展，使你不得不用Erlang重写基础架构，或者用C++重写关键子系统。这才会涉及能力或速度问题。

如果是初涉Web编程，在平台选择上没有坚定的想法，你可能会考虑使用某种动态语言，如Ruby、Perl、Python、JavaScript（Node.js）或PHP。它们相对简单易学，能被Web主机很好的支持，并且有大量的库可用，有助于应用代码的编写。

有微软开发者生态系统使用经验的读者，可能愿意使用C#或ASP.NET。如果你具备与JVM相关的技能，Java或Scala可能是一个好的选择。如果你熟悉函数编程，Clojure、Erlang或Haskell也能完成此工作。

接下来，我们看一个用Perl语言实现的、使用了Dancer框架的后端代码的例子。Dancer这个轻量级的Web框架使用的思维模式与Ruby的Sinatra类似。在这里，我们只介绍最有趣的部分，但你可以访问<https://github.com/osfameron/aBookOfThings-examples/> 查看完整的示例，以及本书讨论的其他代码。（该网址的所有代码都是开源的，你可以自由地创建分支，贡献代码，或许还可以把这些代码用你最喜爱的编程语言重新实现。）

用Perl实现后端代码

就像我们提到过的其他编程语言一样，Perl既有优点也有缺点。它的开发比较活跃，有很好的生态系统，在CPAN上有数量众多的库。它的性能足够强大，如果系统在架构方面做得比较好，则足以以为大型网站提供服务。鉴于Perl是哈基姆最熟悉的编程语言这一事实，在我们这个实例中，不应该高估Perl的缺点（某些地方的语法比较难懂，在一些地区的工作机会不太稳定）。

在少量样板代码之后，程序的主要构成部分是不同API调用的处理程序。每段处理程序都会声明一个HTTP动词（GET，POST，PUT，DELETE）和一个做具体处理的子程序。参数可以在子程序中传递，并且要用分号标记（例如，:id），或者可以把它作为HTTP请求的一部分。

#1 创建一个新的定时任务

```
post "/timers.:format" => sub {
  my $user = require_user;
  # 'require_user' 需要一个会话Cookie
  # 或一个有效的MAC地址

  my $duration = param 'duration'
    or return status_bad_request('No duration passed');

  my $timer = schema->resultset('Timer')->create({
    user_id => $user->id,
    duration => $duration,
    status => '0', # open
  });
  return status_created({
    status=>'ok',
    id => $timer->id,
  });
};
```

#2 改变定时任务的时长

```
put "/timers/:id/duration.:format" => sub {
  my $user = require_user;
  my $duration = param 'duration'
    or return update_complete;
  my $timer = require_open_timer;
  # 如果返回的状态是'0', 则表明定时器是开启的

  ## 注意: 在下面的计算中, 增加的时间是从当前时刻算起
  my $start_datetime = $timer->start_datetime;
  my $new_end_time = DateTime->now->add(
    minutes => $duration );
  my $total_duration = ($new_end_time - $start_datetime)
    ->in_units('minutes');

  $timer->update({ duration => $total_duration });

  return status_ok({
    ok => 1,
    message => 'Timer length updated',
  });
};
```

#3 标记定时器为完成状态

```
put "/timers/:id/complete.:format" => sub {
  my $user = require_user;
  my $timer = require_open_timer;
```

```

    $timer->update({ status => 'C' });

    return status_ok({
        ok => 1,
        message => 'Timer marked complete',
    });
};

#4 取消定时任务
del "/timers/:id.:format" => sub {
    my $user = require_user;
    my $timer = require_timer;

    $timer->update({ status => 'D' });

    return status_ok({
        status => 'ok',
    });
};

#5 为定时任务添加描述信息
put "/timers/:id/description.:format" => sub {
    my $user = require_session;
    # 'require_session' 需要一个会话Cookie!
    my $timer = require_open_timer;
    my $description = param 'description';

    $timer->update({ description => $description });

    return status_ok({
        ok => 1,
        message => 'Description updated',
    });
};

#6 获取定时器列表
get "/timers.:format" => sub {
    my $user = require_session;

    return status_ok({
        status => 'ok',
        timers => [ map $_->serialize, $user->timers ],
    });
};

#7 获取定时器的信息
get "/timers/:id.:format" => sub {
    my $user = require_session;

```

```

    my $timer = require_timer;

    return status_ok({
        status => 'ok',
        timer => $timer->serialize,
    });
};

#8 TODO 设置用户设备的MAC地址

#9 TODO 获取用户设备的MAC地址

#10 登录
post "/login.:format" => sub {
    my $username = param 'user';
    my $password = param 'pass';

    my $user = schema->resultset('User')->find({
        email => $username });

    if ($user && $user->check_password($password)) {
        session user_id => $user->id;
        return status_ok({
            status=>'ok',
            message=>'Login OK',
        });
    }
    else {
        return status_bad_request("Bad username or password");
    }
};

#11 注册用户
post "/user.:format" => sub {
    my $username = param 'user';
    my $password = param 'pass';

    if (schema->resultset('User')->find({ email => $username }))
    {
        return status_bad_request("Duplicate user");
    }
    else {
        my $user = schema->resultset('User')->create({
            email => $username,
            password => $password,
        });
        return status_created({
            status=>'ok',

```

```
        id => $user->id,  
    });  
}  
};
```

请注意，所有的请求都是以`.:format` 结尾。这意味着你既可以把请求递交到`http://api.roomofthings.com/timers.json`，从服务器获得JSON格式的返回结果，也可以递交到`http://api.roomofthings.com/timers.txt`，获得简单字符串形式的返回结果。第二种返回结果的格式针对微控制器做了优化，使其能够更容易地被微控制器解析。

前面这段代码调用了Dancer和Dancer::Plugin::REST中定义的一些函数，如`status_ok` 和`param`。被该代码清单省略的，需要我们编写的其他代码如下。

- 数据库定义（只是用来创建`users` 和`timers` 表），以及使用`DBIx::Class`（一个对象关系映射层，类似于ActiveRecord或LINQ），把这两张表连接到Perl的代码。
- 用于用户管理的几个实用工具函数：`require_user` 和`require_session`（用来对“MAC地址或Cookie”和“Cookie”进行区分）。
- 类似的实用工具函数`require_timer` 和`require_open_timer`，用来从数据库中获取定时器对象。
- 对Dancer/PSGI的基本配置，使得对应用程序的测试和部署都比较容易。

7.2.4 使用CURL进行测试

你在开发API的时候，以及完成开发之后，为了对该API进行测试，或对其进行展示，你需要一个与之交互的手段。为此，你可以同时创建相应的客户端程序（一个Web应用或本地应用程序，或是能使你的物联

网项目与该API建立连接的代码)。但对于本例的情况，在我们开发Clockodillo这个物理装置时，该API早已完成开发了。幸运的是，有很多工具能实现与该API的交互。这其中，curl就是一个非常有用的命令行工具，能用来传送包括HTTP在内的各种数据。

例如，只要调用curl

`http://timer.roomofthings.com/timers.json`，你就可以很容易地发出一个GET请求。但是，该API当然是受登录机制保护的。幸运的是，curl能从容地应对这个问题。curl与一台实现了该API的开发服务器进行交互的例子如下：

```
# -F标记使得curl执行递交（POST）请求的操作
$ curl http://localhost:3000/user.json \
    -F user=hakim -F pass=secret
{
  "status" : "ok",
  "id" : 2
}
```

curl只是生成了一个HTTP请求，并把返回的结果输出到了终端界面上。因为该命令行请求的是一个JSON对象，所以返回的结果也是JSON格式，其中包含status和id两个字典格式的数据。

更多的示例代码如下：

```
# 检查登录时能否拒绝错误的密码
$ curl http://localhost:3000/login.json \
    -F user=hakim -F password=wrong
{
  "error" : "Bad username or password"
}

# 保存登录会话到cookie.jar
$ curl http://localhost:3000/login.json -c cookie.jar \
    -F user=hakim -F pass=secret
{
  "status" : "ok",
  "message" : "Login OK"
}
```

```
# 使用保存的会话Cookie登录，并创建一个定时时长为25分钟的定时器
$ curl http://localhost:3000/timers.json -b cookie.jar \
  -F duration=25
{
  "status" : "ok",
  "id" : 1
}

# 把请求方式改为PUT
$ curl http://localhost:3000/timer/1/duration.json \
  -X PUT -b cookie.jar -F duration=12
{
  "ok" : 1,
  "message" : "Timer length updated"
}

# 获取该定时器的信息
$ curl http://localhost:3000/timers/1.json -b cookie.jar
{
  "status" : "ok",
  "timer" : {
    "start_datetime" : "2012-05-21 19:30:40",
    "status" : "0",
    "id" : 1,
    "user_id" : 1,
    "duration" : 12,
    "description" : null,
    "end_datetime" : null
  }
}

# 删除（取消）该定时器
$ curl http://localhost:3000/timer/1.json \
  -X DELETE -b cookie.jar
{
  "status" : "ok"
}

# 再次获取该定时器的信息（注意定时器被删除后，其当前状态变为'D'）
$ curl -b cookie.jar http://localhost:3000/timers/1.json
{
  "status" : "ok",
  "timer" : {
    "start_datetime" : "2012-05-21 19:30:40",
    "status" : "D",
    "id" : 1,
```



```
        "user_id" : 1,  
        "duration" : 12,  
        "description" : null,  
        "end_datetime" : null  
    }  
}
```

如果你不熟悉代码的话，上面这些示例看起来可能晦涩难懂。但尽管如此，我们希望你能读懂它们，对它们做的事情有所了解。这些代码使用了API中主要的方法，展示了各种基本的完好性测试。通过执行这些测试，你能确保代码的功能符合预期。知道需要与物联网装置交互的Web服务已经基本就绪后，你就可以继续去开发装置上的代码了。

7.2.5 进一步的工作

前面的草案需要再做一些调整才能成为产品级的API。对定时时长的改变不够完善。在用户试图改变定时时长时，对于定时器已经超时的情况，代码没有进行相应的处理。也许定时器的数据结构也应该扩展一下，使得一个定时器能存储更多的历史数据（例如，如果用户多次改变定时时长，服务器能存储每一次的改变量，而不仅仅是记录总时长）。

对于本例中的一些架构方面的特性，我们还没有仔细研究过。

API的调用频率限制

如果服务广受欢迎，对网站的连接次数进行管理就变得至关重要了。对每天、每小时或每分钟的最大调用次数进行设定可能是有用的。为了做到这一点，你可以为每个你想施加限制的时间段设置一个计数器。在这之后，身份验证过程只需简单的增加计数值，如果计数值超过了一个预定义的阈值，则验证不成功。使用在预定时间执行的cron定时任务，可以把计数器重置为0。

虽然一个软件应用能很容易地提醒用户，他们的调用次数已经超过了限制值，应该稍后再试。但如果一个物理装置突然被限制使用，用户

可能会认为它坏掉了。解决这一问题的可能的办法之一，就是根本不对来自于装置的调用做次数限制。

把OAuth用于与其他服务之间的身份验证

尽管OAuth（目前）可能不是最好的用于连接微控制器的解决方案（当前还没有适用于Arduino的OAuth库），但后端服务没有理由不接纳OAuth，用它来连接Twitter、音乐发现网站last.fm或Web自动化网站IFTTT（If This Then That）等服务。

经由HTML交互

该API当前只能序列化成JSON、XML和文本格式的输出。你可能也想用Web浏览器连接服务。我们在最初考虑API的设计时，把任务分解为两部分：装置能完成的任务和其他任务。后者可以很容易地在基于浏览器的应用中实现。当然，用户不会愿意使用原始的API调用，并且执行操作的流程可能会稍有不同，但被操纵的基本数据是相同的：我们已经介绍过的调用将构成Web应用的核心，正如它们在物理装置中所起的作用那样。

需要注意的是，并不是每个物联网产品都需要浏览器应用。也许API就可以满足你的全部需要。可能还需要一个静态主页，其中包含一些关于怎样调用API的文档。

但就Clockodillo而言，我们的确需要一组网页，用来与服务交互：用户应当能够查看他们的定时器，指定描述信息，等等。

只为HTML应用设置特定的处理程序很容易，例如：

```
post '/login.html' => sub { ... }
```

不过，使用与之前完全相同的代码，只是增加一个HTML格式选项可能更加优雅。这段代码可以把数据注入一个HTML模版，而不是返回一个JSON字符串。例如，只需访问/timers/1234.html而不

是/timers/1234.json，就能在稍后获得一个针对人而不是物联网装置的定时器数据视图。

缺点

尽管Web浏览器的确支持HTTP协议，但它们在通信时，通常不能支持我们讨论过的全部方法。特别是，它们往往只支持下列方法。

- **GET**：用来打开网页，以及在点击超链接时打开其他网页。你可以链接到<http://timer.roomofthings.com/timers/1234.html> 页面，获取HTML版本的API调用结果（使用get_timer模版）。
- **POST**：用来提交表单或上传文件。为了递交定时器的信息，你可以创建一个如下的Web表单：

```
<form method="POST" action="/timers.html">
  <input type="text" name="duration">
  <input type="submit" value="Create a new timer!">
</form>
```

该表调用用了**POST** 方法，并返回适当的HTML页面。

但对于精心设计的**PUT** 和**DELETE** 方法，情况又怎样呢？Web浏览器通常不支持它们。不过不用担心，JavaScript支持这些方法。解决这个问题的一种办法是在JavaScript中调用它们。另一种办法是采用“隧道”方式，把请求封装到**POST** 方法中。在Perl中，规定使用**x-tunneled-method** 这个字段实现该功能。你可以这样实现：

```
<form method="POST"
  action="/timer.html?x-tunneled-method=DELETE">
  <input type="hidden" name="id" value="1234">
  <input type="submit" value="Cancel this timer!">
</form>
```

现在你只需要设法让Web框架接受这个**POST** 请求，并且把它当作**DELETE** 请求。对于采用Dancer的示例应用，我们使用了模块 `Plack::Middleware::MethodOverride`，并且只用一行代码就实现了此功能（<https://metacpan.org/module/Plack::Middleware::MethodOverride>）。其他Web框架也有类似的扩展模块。

或者你也可以不使用Web浏览器，而是使用完全不同的代码库编写Web应用，并且通过API与主服务交互。这是一种有吸引力的组合，因为它迫使针对人和针对装置编写的代码使用（操作）相同的API，从而增加了对API的测试次数，避免忽略针对装置编写的代码。是否决定采用这种做法，在很大程度上要依赖于你的团队的技能组合。

设计适合人类使用的Web应用

不管你选择用什么方法实现Web应用，以及本章花费大部分篇幅介绍的基于文本的API，你都能很容易地获得一个优雅的、设计良好的应用，用来实现与人的交互。

因为有大量关于设计Web应用的优秀书籍，我们不会讨论这方面的任何细节，但你可能会有兴趣看一些例子，以借此思考一下设计的过程。

例如，图7-2显示的是一个静态的登录页面，是**GET** 请求的返回结果。该API调用甚至没有指明**GET**方法，这样做对计算机来说好像是多余的。这个页面完全是为了方便人类查看而设计的。所有的标签，如“你的email地址”，以及帮助文本，如“别忘了你的密码区分大小写的”，完全是为了给用户提供一些指引。出现在页面中的Clockodillo标识，除了能证明我们真的不是设计师外，也展现了网站的品牌和外观。

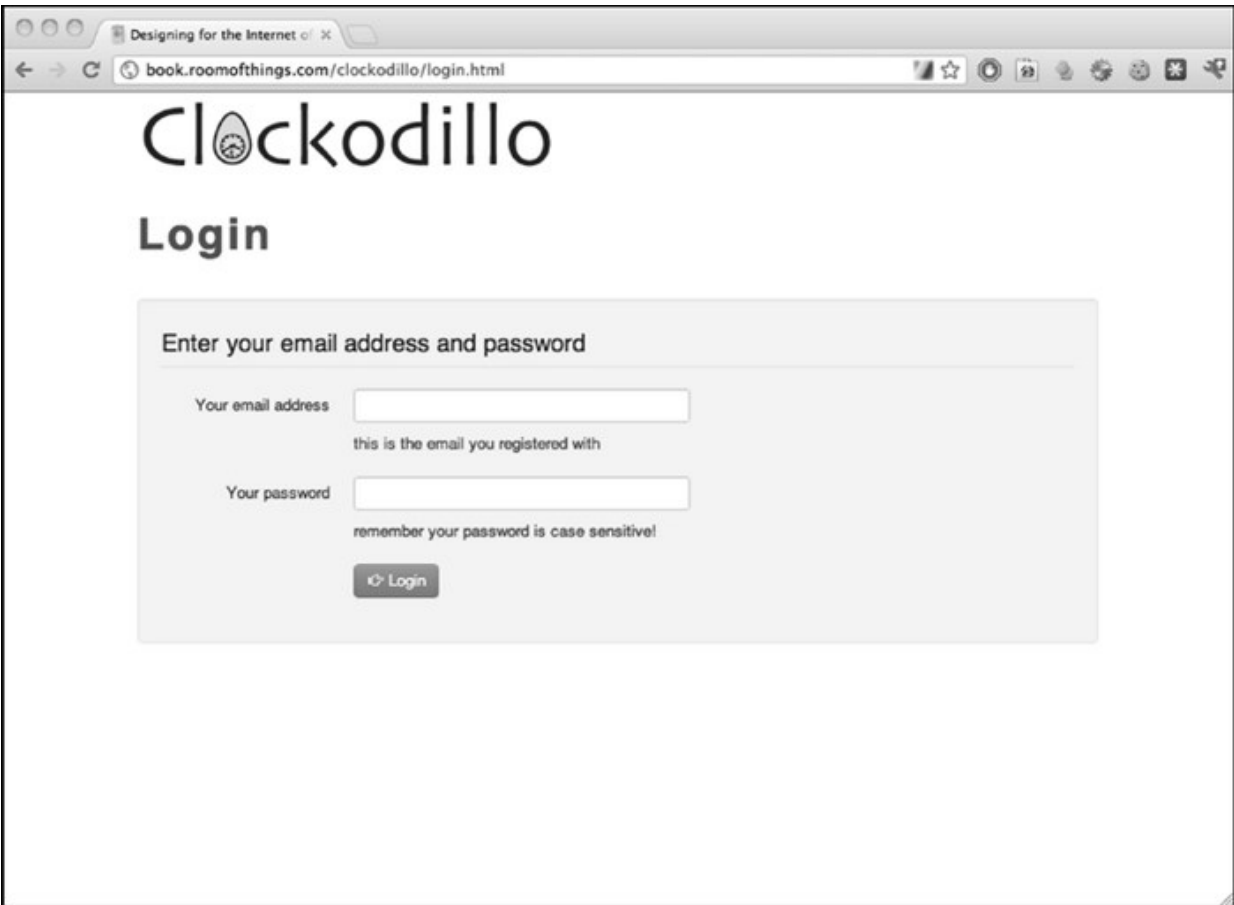


图 7-2 适合人类查看的Clockodillo登录页面

图7-2只是一个简单的例子。图7-3显示了一个极端的改变。定时器列表的呈现形式，不是一个包含原始数据结构的JSON字符串，而是一个经过高度优化的格式。日期字段已格式化为适合人类查看的格式。定时器的定时时长和状态（正在定时、完成、放弃）通过对不同颜色、进度条和定时时长标识的运用，实现了可视化的表示。该页面还链接到了其他操作：**Edit**按钮能打开一个页面，允许做一些其他操作，如改变描述信息等；页面上方的菜单栏链接到了其他功能，用来帮助用户在他们想执行的任务间切换。



图 7-3 适合人类查看的定时器列表

最后，在设计页面的过程中，我们增加了一个输入框，用来搜索某个特定的定时器。在为API的设计做准备时，我们甚至不会想到这个功能（定时装置不需要这个功能）。一旦从人类用户的视角考虑问题时，这个功能显然是需要的。从两个截然不同的视角（机器和人）审视你的产品，将会使产品的设计变得更好、更稳固。

7.3 实时响应

我们已经介绍了一类传统形式的API。使用此类API时，你向服务器发送HTTP请求，就能从服务器接收响应。如果你需要一个反应非常敏捷的系统，采用这种方式并不太好。建立一个HTTP请求需要在本机和服务器之间进行几个来回的交互，这一过程通常被称为“三次握手”，它包括：客户端发送一个SYN（同步）请求，服务器发送SYN-ACK确认请求，最后客户端再发送一个ACK。尽管这一过程几乎可以在瞬间完成，但有时也可能会花费较长时间。

建立连接所花费的时间可能要紧，也可能无关紧要。任何功能很强大的单板都能在后台执行建立连接的操作，并在连接建立好后做出响应。对于诸如Arduino这样的准系统板，当前的以太网/HTTP盾板和相关的库往往会在建立HTTP连接时阻塞其他操作。这意味着在这段时间内，微控制器难以做其他处理（尽管可以使用硬件中断迂回地解决这一问题，但这样做会产生一些限制并且会使问题复杂化）。因为建立连接的过程通常发生在一块配备有处理器的附加电路板上，能够并行地处理这件事，从而避免阻塞主线程，所以在将来这个限制很可能会被解除。

如果你想在单板上发生某件事件的瞬间执行一项操作，你可能需要将建立连接的时间考虑在内。如果服务器必须马上执行一项操作，根据建立连接所需时间的不同，这个“马上”可能是将近一分钟以后。以任务定时装置为例，你可能想把用户手离开刻度盘的瞬时登记为准确的开始时间，但你实际登记的时间包括了建立连接的时间。

我们将在这里介绍两种技术选择：轮询和所谓的Comet技术。之后，在关于非HTTP协议的小节中，将介绍MQTT、XMPP和CoAP等其他可选的解决方案。

7.3.1 轮询

如果想让装置或另一个客户端能即刻作出响应，该怎么做？不知道想要作出响应的事件的发生时间，因此也就无法在数据恰巧生成时发出请求。考虑下列两种情况：

- 用户在办公室签到瞬间，WhereDial应该开始转向“工作”位置；
- 在任务定时器启动瞬间，用户计算机上的客户端程序应该能作出响应，提供键入任务描述信息的机会。

使用HTTP API请求处理上述情况的传统方法是以固定的时间间隔发送请求。这被称为轮询。例如，你可以每分钟执行一次调用，用来检查是否有新数据可用。不过，这也意味着在轮询的结果返回之前，你无法做出响应。因此，对于本例的情况，加上建立HTTP连接的时间，延迟时间可以达到1分钟。你也可以缩短轮询的间隔时间，如改为10秒，但这会给下列装置带来负担。

- 服务器：如果有成千上万的装置连接到服务器，并且每一个装置都周期性地做轮询操作，你需要对系统进行扩展，以适应系统的负载；
- 客户端：如果客户端像之前提到的Arduino的例子，即微控制器在每次连网时都会处于阻塞状态，则这种情况尤为重要。

7.3.2 COMET

Comet是一组用来解决轮询时低效率问题的技术的总称。与很多其他的技术一样，Comet中的很多技术在Comet这个“品牌”出现之前就已经存在了。不过，用一个名称来表述想法是有用的，有助于讨论并交流想法，推动技术向前发展。

长轮询（单向）

第一个重要的开发成果是“长轮询”。客户端像往常一样发起一个轮询请求。对于正常的轮询请求，服务器会立即给出一个响应结果，即便这个结果是“没有要报告的内容”。然而对于长轮询的情况，服务器要等到有数据需要报告时才会返回结果。这意味着服务器必须周期性地向客户端发送keep-alive消息，以防止物联网装置或Web页面认为服务器超时无响应。

长轮询对于WhereDial是一个理想的选择。装置发起请求，要求了解用户位置下一次改变的时间。一旦WhereDial收到返回结果，它就把转盘

转到新的位置，并发起一个新的长轮询请求。当然，如果连接中断（例如，服务器停止发送keep-alive消息），客户端也能发起新的请求。

然而，对于任务定时装置，这种情况不是很典型。你可能想快速地从定时装置发送消息，也想从服务器接收这些消息。尽管你可以给服务器发送消息，但需要为此先建立连接。因此你可以认为长轮询是单向通信。

多部分XHR（MXHR）（单向）

构建Web应用时，经常会使用名为XMLHttpRequest（XHR）的JavaScript API与服务器通信，而无需加载全新的页面。从Web服务器的角度看，这些请求和任何其他HTTP请求没什么不同。但因为预期的接收者是一些客户端代码，所以专门的协议和支持库（包括客户端和服务端侧）已经被开发出来，用来处理这种交互方式。

很多浏览器都支持multipart/x-mixed-replace 这种内容类型。它允许服务器使用XHR发送一个文档连续出现的多个版本。需要注意的是，XMLHttpRequest有些用词不当，因为根本没有规定需要实际使用XML。如果希望能够从服务器接收多个消息，使用XML这种内容类型也许会更加复杂。

你也完全可以仅使用长轮询，每当上一次的请求返回后，马上创建一个新的请求。但这也意味着你可能会漏掉连接建立过程中出现的信息。对于WhereDial，这种情况不大可能发生，因为你不可能把位置更新为“家”，然后又紧接着把位置改变为工作场所。然而，对于某些物联网装置，如艾德里安实时显示提要状态的Xively虚拟仪表，能对服务器端的变化几乎立即做出响应是使用这些装置的根本目的。

HTML5 WebSockets（双向）

在第3章中你已经看到，Web服务使用的HTTP协议是建立在TCP协议之上的。用来直接与TCP层交换信息的API通常被称为套接字API。于是，当Web社区试图基于HTTP层提供类似功能时，他们把这个解决方案称为WebSockets。

尽管WebSockets现在还只是HTML5标准中的一个工作草案，但它看似已经受到最新的浏览器、服务器和其他客户端程序的青睐。

例如，在Arduino平台上已经有WebSockets的一个（部分的）实现（<https://github.com/krohling/ArduinoWebsocketClient>）。

WebSockets的优势在于可以双向通信。你可以把它看作完整的Unix套接字句柄。客户端可以写入请求到WebSockets，也可以从WebSockets读取响应。

对于任务定时装置来说，WebSockets很可能是理想的技术选择。在一个套接字建立之后，定时器可以简单地用它发送信息，包括任务的开始、修改和取消，也可以用它读取信息（用软件所做的修改）。

因为WebSockets是新生事物，并把HTTP协议推向一个稍微非正统的方向，它和代理服务器之间存在一些已知的问题。当前存在这方面问题的代理服务器经过修正后，能支持WebSockets，因此，随着这个问题的解决，情况会发生改变。这可能是一个与系统架构有关的问题。请参阅后面的“扩展性”小节。

实现

前面小节描述的几种技术选择是目前最受青睐的。然而，作为一个正快速变化、没有绝对共识的领域，传输方式的实际细节以及各种限制条件都注定会改变，因此值得在其发展过程中去关注它们。Comet的维基百科页面（[http://en.wikipedia.org/wiki/Comet_\(programming\)](http://en.wikipedia.org/wiki/Comet_(programming))）能有效跟踪了解技术现状。

对于物联网应用，你可能需要考虑的三个主要的部分是：浏览器Web应用（如果适用的话）、微控制器本身，以及服务器应用。让我们看一下各部分对这些技术的支持情况。

在浏览器端，你通常可以对实际的传输方式进行抽象并用一个库实现。而这个库能选择连接服务器的方式。例如，如果WebSockets可用，它可能会使用WebSockets；否则，它会转而使用MXHR或长轮询。这项功能非常有用，因为在现阶段，每种Web浏览器对不同技术都有不同程度的支持。jQuery和Dojo都有广为人知的Comet库可用。

此外，很多Web服务器具有支持Comet技术的抽象。Web::Hippie::Pipe为Perl实现的Web服务器，如Twiggy，提供了一个统一的双向抽象。如果WebSockets可用，还是使用WebSockets，否则会转而使用MXHR或长轮询。你可以为node.js（JavaScript）、Thin（Rails）、jetty（Java）等找到类似的抽象。

在微控制器平台上也有各种可用的库。不过，它们往往只支持一种传输方案。例如，在Arduino上有几个专为该平台实现的WebSockets库可用。在Arduino平台上，实际上不需要回退到一种不同的交换数据的方式。不同于桌面Web应用的情况，使用Arduino时，你无需担心用户会使用不同的浏览器，因为装置的固件是你提供的。

扩展性

一个重要的考虑是，所有这些Comet技术需要在客户端和服务端之间保持一个长期的连接。对于单一客户端，这是微不足道。但如果有很多客户端，服务器必须与每个客户端保持连接。如果你运行的服务器应用开启了多个线程或进程，你实际上是为每一个客户端分配了一个服务器的实例。由于每个线程或进程都会消耗系统资源，例如内存，这种方式不能支持太多的客户端。

作为替代方案，你可能想使用异步Web服务器。服务器应用依次扫描每个客户端连接，在有新的输入或输出时为其提供服务。如果服务器能快速地为每个客户端提供服务，这种方式就能很容易地扩展到支持数万客户端。对于一个典型的Unix服务器，套接字的数量有最大值的限制，所以你能同时连接的客户端数目也是这个最大值。这虽然是一个问题，但显然也是一个好事情。当你的应用达到这个限制时，你就该考虑负载均衡等其他技术了。一个优秀的系统团队将能应用这些技术，依据负载情况实现规模的扩展。

你还可以使用前端代理（Varnish或类似软件）来做一些与持久化的客户端连接相关的处理工作。

7.4 其他协议

如你所见，HTTP是因特网上非常流行的协议，但它也不是完全适用于所有情形。如果你能掌控连接的双方，你可以选择使用一种完全不同的协议，而不是使用前面介绍的某种解决方案迂回地应对HTTP的局限性。

虽然有很多协议可供选择，但我们只简要介绍其中几个较适用于物联网应用的协议。

7.4.1 消息队列遥测传输

消息队列遥测传输（MQTT，<http://mqtt.org>）是一种轻量级的消息传输协议，专门针对网络带宽受限或代码的内存占用量受限的应用场景设计而成。它最初由IBM开发，但已作为一个开放的标准发布。它目前有若干种实现，开源和闭源方式的都有，还有很多针对不同语言的库可用。

不同于HTTP采用的客户端-服务器模型，MQTT使用了发布/订阅机制，实现了经由消息代理的消息交换。发送者把消息发布到消息代理上的某个特定主题，而不是把消息发送给一组预定义的接收者。接收者订阅那些它们感兴趣的主体。一旦有新的属于该主题的消息被发布，消息代理会把这个消息发送给所有感兴趣的接收者。这使得实现一对多的消息传输更加容易，并消除存在于HTTP中的客户端和服务端之间的紧密耦合。

MQTT有一个用在传感器上的姐妹协议，即MQTT-S。该协议适用于TCP不可用且资源非常受限的平台或网络环境，令MQTT的应用领域扩展到诸如ZigBee之类的传感器网络。

7.4.2 可扩展通信和表示协议

另一个消息传输的解决方案是可扩展通讯和表示协议（XMPP，<http://xmpp.org>）。XMPP是由Jabber即时消息系统发展而来的，因此作为因特网上的一个通用协议，它获得了广泛的支持。它好坏参半：

已被很好理解，并获得了广泛部署，但由于不是明确地专为嵌入式应用而设计，它采用了XML作为消息格式。选用XML会令消息相对冗长，可能不太适用于内存受限的微控制器。

7.4.3 受限应用协议

设计受限应用协议（CoAP）旨在解决适用于HTTP的同类问题。与MQTT-S类似，它也被用于没有TCP的网络。现有的提案包括：在UDP之上和手机短消息系统之上运行CoAP，CoAP与6LoWPAN集成。

CoAP吸取了很多HTTP的设计特性，具备一个预定义的连接代理的机制，从而允许从一种协议映射到另一种协议。在写作本书的时候，该协议正经历成为一个建议标准的最后阶段，相关的工作由IETF（因特网工程任务组）的CoRE（Constrained RESTful Environments）工作组负责协调。

7.5 小结

本章较详细地介绍了物联网的网络端。我们先介绍了如何通过公开的API或Web数据抓取与现有的服务进行交互。然后通过一个实例，展示了在有需要时，怎样创建全新的API。

连同之前的两章，你现在对构建物联网装置原型所需的工作的广度应该有较好的体会了。把原型转化为产品还有更多工作要做，这些将在后续章节中介绍。如果你只打算构建一个让自己的生活更加轻松或有趣的物品，你应该已经准备就绪，可以即刻开始行动了。

在下一章，我们将返回到物联网装置，对你所需的嵌入式系统的编程技术做更详细的探讨。下一章的内容介绍了嵌入式编程与标准的台式机或服务器编程的不同之处，并就如何处理这些不同以及出现意外情况时如何调试给出了建议。

第8章 嵌入式编程技术

在大多数情况下，为嵌入式平台编写代码和为桌面或服务器系统编写代码没有什么不同。然而它们之间确实存在一些差异，在你编写代码时，值得将其牢记于心。在本章中，我们将探究一下其中的一些问题，并且提出一些方法，使你能避免或迂回地解决这些问题。

如你在第5章中所见的那样，嵌入式系统和“标准的”计算平台之间最大的差别之一是可用资源的匮乏。虽然笔记本电脑或服务器上配备了几千兆字节的内存，也有成百上千兆字节的存储器，但微控制器的资源通常以KB为单位。例如，Web浏览器可以一次性地将330KB的HTML、CSS、JavaScript代码和图像接收到内存中，然后再将其复制到各个地方，对它进行解析，转化为更适合显示的数据结构，以显示一个（相对简单的）谷歌主页。这么做对浏览器来说没什么，但330KB差不多是Arduino Uno板上全部可用内存¹的150倍。而这还只是Web浏览器下载资源所需的内存，尚未对其进行任何处理。

¹ 这里的内存应该指的是SRAM，Arduino Uno所使用的ATmega328芯片只有2KB的SRAM。
——译者注

除了硬件资源限制，联网装置可能生来就是一种开启后即被遗忘的物品。当然，不会真的被忘记，因为人们还指望它们提供有价值的服务或者给生活带来快乐呢。然而，装置的所有者并不希望对其进行周期性的重启或维护。因此，你的系统需要在没有任何用户干预的情况下，一次运行数月或数年。

这条原则也适用于对系统的配置和调整。对于服务器软件来说，通常会由一名系统管理员留意各种状况并且不时地对其进行一些维护，虽然这种作法大体上是可以接受的，但对于笔记本电脑和PC机之类的设备，这种做法却并不合适。例如，作为明确由最终用户负责的工作，磁盘碎片整理之类的任务的重要性正日渐降低。对于普适计算装置而言，它的目标应该是更进一步地体现自动或自我维护的观念。

8.1 内存管理

如果你没有很多内存可用，在使用内存时就需要精打细算，尤其是当你没办法将内存使用情况告知用户的时候。计算机用户看到太多“内存不足”的警告对话框，会尝试重新启动。同样，系统管理员发现服务器正在折腾硬盘，把内存中的页面扩展到硬盘，从而增加虚拟内存的容量时，也会重启服务器。相反，没有屏幕或其他指示器的嵌入式平台会盲目地继续运行，直到内存完全耗尽为止。此时，它通常会不知原因地停止运作，以此来向用户“指示”当前的状况。

对于资源受限的装置，即便在你为其开发软件时，对这些问题进行调试也是非常困难的。一分钟前工作良好的装置却莫名其妙地停止了工作。唯一的差异可能是调试日志中多了一个难以察觉的字符。更糟的情况下，差别会非常细微，例如，循环过程只是被多执行了几次。

8.1.1 内存类型

在介绍如何充分利用可用资源的具体细节前，我们需要对你可能会接触到的不同类型的内存做些解释说明。

ROM

只读存储器（**ROM**）指的是在芯片的制造阶段，以硬编码的方式存储信息的内存，并且在此之后只能读取这些信息。这种内存最不灵活，通常只用来存储可执行程序代码和固定的、从不改变的数据。最初使用**ROM**的原因是，它是最廉价的制作内存的方式。但如今，**ROM**与闪存芯片相比并无成本优势，而闪存具有更大的灵活性，所以纯**ROM**芯片几乎要绝迹了。

闪存

闪存是一种半永久性的内存类型。它具备**ROM**的全部优点，即无需任何供电也能存储信息，因而其中的内容不会因为电路失电而丢失。同时，它没有不可改写内容的缺点。虽然闪存的内容可被改写的次数是一定的，但在实际使用中很少能达到这种限制。闪存的读取速度和**ROM**或**RAM**比，没有太大差别。然而，写入闪存时需要几个处理器周期的时

间。这意味着闪存最适合存储你想保持不变的信息，如程序的可执行文件本身或已收集到的重要数据。

RAM

随机存取存储器（**RAM**）通过牺牲持久性换取存取速度的提升。需要通过供电保持它的内容，但它的写入速度和读取速度（特别是与闪存相比）是一致的。因此，它被用作系统的工作内存，即用来存储正在处理中的数据。

系统拥有的持久性存储空间往往要比**RAM**大很多，所以把尽可能多的内容放置在闪存中是合理的做法。显然，程序代码本身驻留在闪存中。你也可以给编译器（负责把源代码转换为处理器能理解的机器码的程序）提供一些提示，以帮助其将正在运行的程序尽可能多地放置到闪存中。

如果一个变量的内容不会改变，最好是将其定义为一个常量。在C和C++语言（常用于嵌入式系统）中，使用**const** 关键字定义常量。通过该关键字，编译器能够了解到这种变量不需要驻留在**RAM**中，因为它不会被改写，只用来读取。如果有任何大的查找表或其他大的数据结构，如字体或位图，采用这种定义常量的方式能节省很多**RAM**。即便是用于调试目的的文本字符串也能占据数量可观的**RAM**空间，因此最好将它们从**RAM**转移到闪存中。

对于某些处理器架构（根据笔者的经验，**Arduino**使用的Atmel芯片所采用的哈佛架构是最明显的例子），数据（**RAM**）和程序（闪存/**ROM**）所在的内存空间是分开的。这意味着在两者之间互相交换数据不是太容易。因此，当你想使用闪存中的数据时，你可能不得不做一些额外的工作，把数据从闪存复制到**RAM**中。你通常能找到经得起实践检验的方法来做这件事，但你需要多做一点工作，而不仅仅是把字符串和其他大的变量声明为常量。当你在这些平台上工作时，要意识到这个问题的存在。

例如，**Arduino**平台提供了一个额外的宏定义，用来指明某些字符串应该被存放到闪存，而不是**RAM**中。通过用**F(...)** 包裹字符串的方式，可以告诉系统：这是一个“闪存”字符串，而非一个常规的字符串：

```
Serial.println("This string will be stored in RAM");  
Serial.println(F("This one will be in flash"));
```

8.1.2 最大程度地利用RAM

现在，你已经把能从RAM中移走的内容都移到闪存中了，剩下要做的就是想办法更好地对空闲的RAM进行利用。

当只有几千字节或几百字节的RAM可用时，很容易会把内存用光，导致装置运行异常或崩溃。然而，为了提供更多的特性，你可能想使用尽可能多的内存。这种考虑非常重要。如果内存使用量是确定的，也就是说，你知道将被使用的内存的最大值，则很容易在RAM使用的最大化和可靠性之间找到最佳的折衷方案。

实现这点的方案就是，不要在程序运行的时候动态分配内存。对于有大型系统编程经验的人来说，这个观点有点奇怪。例如，当从因特网下载某些信息时，究竟怎样才可能事先确切地知道它将有多大？如果你要提取的网页在编写完代码之后，又额外增加了一两个段落，会发生什么情况？你的算法必须考虑到这种可能性。当你知道需要多少内存时，在台式机或服务器系统上，标准的做法是在下载时就分配足够多的内存。

在一个确定性内存模型中，需要采取不同的策略。不能采取为整个网页都分配空间的做法。除了预留空间以存储将提取的重要信息之外，还需要分配一个内存缓冲区，把它用作下载或处理网页时的工作区。网页不是一次性地被整个下载到内存中，而是被分块下载的，即每次都是先把缓冲区填满，然后对这个数据块进行处理，在这之后再下载下一个数据块。在某些情况下，你可能需要在下载下一个数据块之前记住当前块的一些内容，例如，你正在解析的网页的一个关键部分跨越了数据块之间的边界。不过，一个精心设计的算法通常能解决这样的问题。

这种方法的优点是，你能处理的网页的大小要比不采用这种方法时大很多。也就是说，你可以处理比系统全部可用内存都大的数据集。虽然Bubblino的代码在仅有2KB RAM的Arduino板上运行，但它可以很容易地处理来自于Twitter搜索API、每次下载量通常在10~15KB之间的标准XML格式的响应。由于Bubblino的代码把所有的文本浓缩为一个代表新推文数量的数字，它可以在处理过程中丢弃大量数据。它需要跟踪的只是每条推文是否是新发布的，以及它所看到的最新推文的时间标签（下一次它就能从该时刻开始计数）。

然而，这种单次解析的缺点是，你没有办法对数据流进行回溯。正在处理的数据块被丢弃后就消失了。如果正在使用的数据格式，使得直到对文件后面的部分进行解析时，才能知道是否需要某个数据，那么你不得不留出空间，在碰到可能有用的数据段时把它保存起来。这样，当解析到可以做决策的地方时，仍然可以使用这个数据。此时如果认为这个数据不被需要，就可以丢弃它。

使用这种类型的解析，也意味着在处理数据之前，通常无法建立复杂的数据结构，无法检查数据是否正确或者完整。例如，你往往受条件所限，不能对XML文件进行严格解析，只能做比较基本的完好性测试，即看它的格式是否良好（看起来像是一个XML文件），而不是看是否有效（符合给定的schema）。如果能做额外的复杂检查对于系统而言是一项重要的功能，那么就需要考虑其他替代办法了。在这种情况下，可以首先选择有更多可用内存的系统。或者，把下载内容缓存到SD卡或闪存的其他区域。这样就可以对这些内容进行多次处理，而无需把它们一次性地全部读到RAM中。

管理RAM：栈与堆

系统刚启动时，它的所有RAM都能用来存储内容，但系统如何决定把各种内容放到什么位置，之后又怎样找回呢？这里有两个和内存安排有关的通用概念：栈和堆。栈和堆各有优缺点，在计算机（包括大多数嵌入式系统）中往往都会被用到。

正如其名称所暗示的，栈的组织方式就像一叠文件。添加到栈上的新数据项被放置在最上面，并且只能严格地按相反次序移除数据项。因此，第一个被移除的数据项正是最晚添加到栈上的数据项。

这种安排方式使得处理器可以容易地跟踪各种内容的位置，了解已使用了多少内存空间，因为它只要对栈顶进行跟踪就可以了。使用栈的缺点是，如果某个特定的变量已不再被使用，只能等到可以将其从栈中移除时，才能释放它所占用的内存。即只有当在它之后被添加到栈中的数据项都被移除后，该变量才能被移除。

因此，栈只对下列情形是真正有用的：

- 生命期不太长的数据项；

- 在整个程序的生命期中，不断被用到的数据项。

始终可用的全局变量会最先在栈上获得分配的空间。在这之后，每当执行路径进入一个函数，该函数中声明的变量就会被添加到栈上。该函数的参数会被立刻推送到栈上，而其他变量会在执行流程到达时入栈。因为函数中所有的变量只能被函数中的代码使用，当执行流程到达该函数的结尾时，所有这些参数和变量都可以被丢弃了。这样就可以对栈做反向操作，让它的大小恢复到执行流程进入函数之前的状态。

随着算法的流程进入嵌套层次较深的函数，栈空间的占用量会增加。而随着执行流程从这些函数返回，这些占用的空间会被释放。如下面这段伪码：

```
// 全局变量
function A {
    variable A1
    variable A2
    call B()
}
function B {
    variable B1
    variable B2
    variable B3
    call C()
    call D()
}
function C {
    variable C1
    // 做一些处理
}
function D {
    variable D1
    variable D2
    // 做一些其他处理
}

// 主执行流程从这里开始
// 调用函数A做某事...
call A()
...
```

栈空间的使用情况如下：

- (1) 在调用函数A之前，栈的占用情况如状态（i）所示；
- (2) 随着执行流程进入函数A，函数A的变量被加入栈（ii）；
- (3) 之后在函数A中调用函数B，导致函数B的变量被加入栈（iii）；
- (4) 在函数B中，先是函数C被调用，导致函数C的变量被加入栈（iv）；
- (5) 在执行流程从函数C返回后，函数C的变量被从栈中移除，恢复到状态iii；
- (6) 之后函数D被调用，于是函数D的变量被推送到栈上（v）；
- (7) 执行流程返回函数B，函数D的变量被从栈中移除（iii）；
- (8) 执行流程返回函数A，函数B的变量被从栈中移除（ii）；
- (9) 最后，在执行流程离开函数A后，栈中只剩已定义的全局变量（i）。

如你所见，栈空间上的最大内存使用量在很大程度上取决于代码的执行路径。

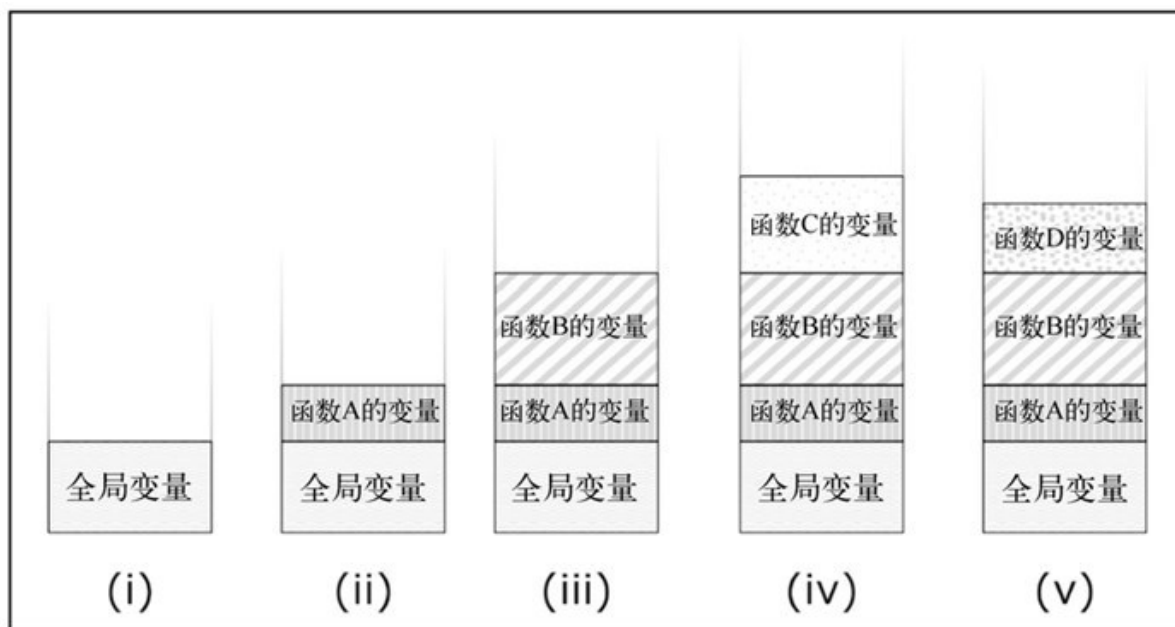


图 8-1 示例程序执行流程中各点的栈使用情况

相比之下，堆允许你在任何时候分配内存块，并且占用多长时间都可以。

堆有点像火车的座位区，严格地按照从前到后的顺序安排座位，并且让组团出行的乘客挨着坐。开始的时候，所有的座位都是空着的。随着各个组团的乘客到达，就需要把他们引领到下一片空闲的座位区。

举例来说，如果一组乘客有6个人，他们在某一站下车，火车座位区的中部就会空出6个相邻的座位。如果下一组上车的乘客有3个人，他们可以占据空座中的一半，剩下3个相邻的空座。

在下一站，又有一组乘客上车，他们有4个人。因为他们要坐在一起，3个空座坐不下，所以需要让他们坐到火车的后部。差不多所有的空座位都在这里。

随着旅客的到来和离开，座位的占用和腾空，此类行为会愉快地持续下去。但可能存在两种问题。首先，可能会发生人多座少的情况（和内存耗尽是相同的问题）。第二个问题更微妙：尽管理论上有足够多的空座可以提供给下一组乘客，但这些空座位分布在火车上

的各个区域，没有连续的空座区可用。后面这种情形被称为内存碎片化。

正如我们在介绍栈时所做的那样，使用一些伪码将有助于演示堆的正常使用情况：

```
create object A (size 20 bytes)
create object B (size 35 bytes)
create object C (size 50 bytes)
// 做一些需要用到对象C的工作
delete object C

create object D (size 18 bytes)

// 再做一些与对象B和D有关的工作
delete object B
create object E (size 22 bytes)
```

随着代码执行流程的推进，堆将按以下方式演变：

- (1) 在执行流程开始时，堆是空的 (i) ；
- (2) 对象A被加入堆 (ii) ， 占据20字节的空间；
- (3) 对象B被加入堆 (iii) ， 其位置就在对象A之后，再用掉35字节；
- (4) 对象C被加入堆 (iv) ， 其位置就在对象B之后，堆的大小增加了50字节；
- (5) 对象C不再需要，被删除。对象C所占用的堆空间被释放，堆的大小恢复到状态iii；
- (6) 对象D被创建，在对象C刚腾空的位置，占用了18字节的空间 (v) ；
- (7) 对象B不再使用，被删除。因为其他代码可能会依赖于对象D的位置，不能移动对象D，因此，在对象A和D之间，现在出现了一段空闲空间 (vi) ；

(8) 对象E被创建。它需要22字节的空间，这意味着它将被放入对象B腾空的空间（vii）。

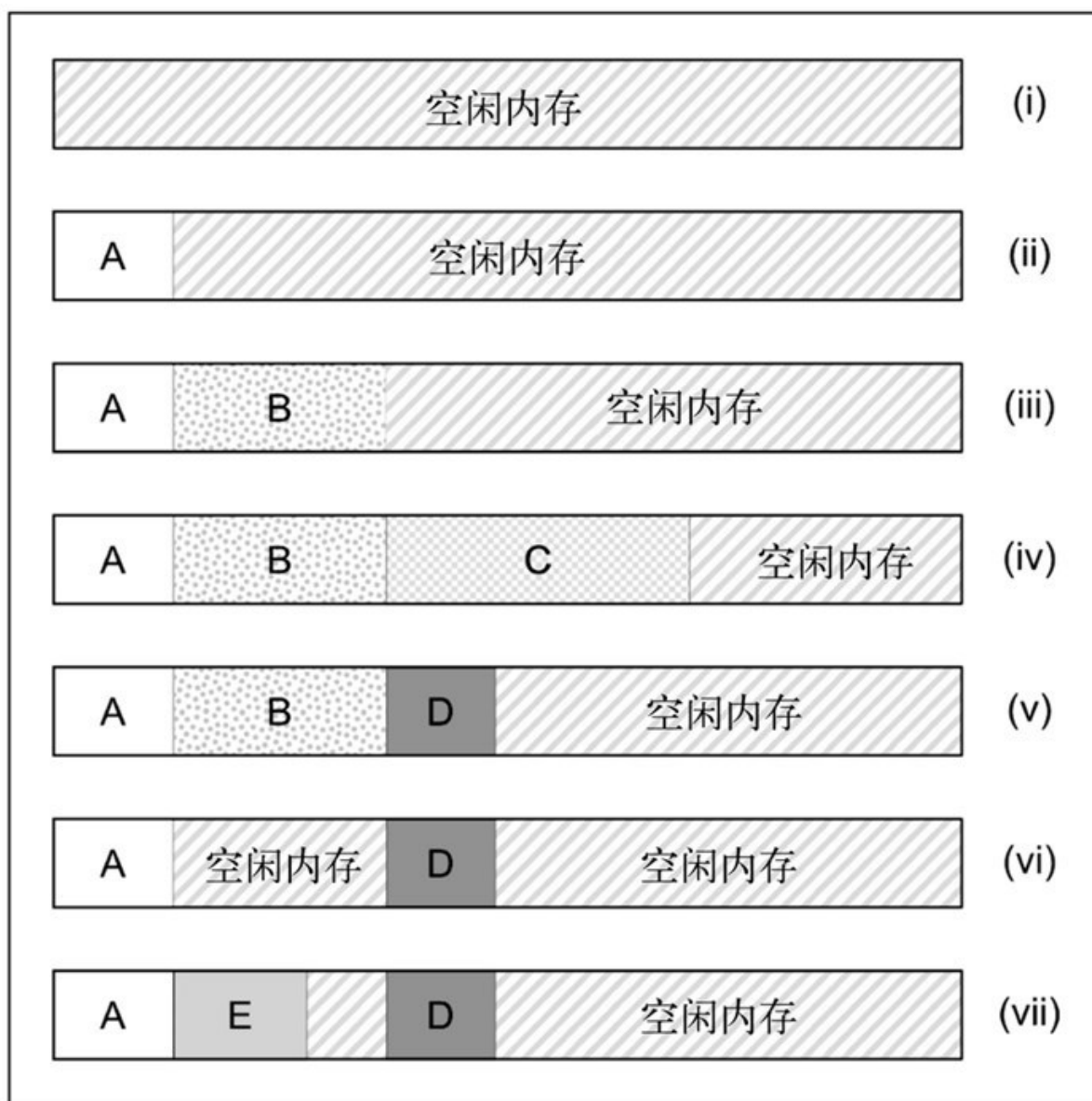


图 8-2 示例程序执行流程中各点的堆使用情况

所谓确定性内存使用，可以归结为：无论如何，避免使用堆。也就是说，对于只有几千字节RAM的系统，我们建议只使用栈来存储变量。但严格来说，只使用栈的还是有可能耗尽内存。这种情况被称为栈溢出。在一些处理器上，发生这种情况是由于栈是内存中划分出来的一个大小固定的区块。如果程序在执行流程中的某些位置需要的栈空间超出了这个区块的大小（例如，进入了一个深度嵌套的递归过程），那么栈空间

就会超出边界。对于某些体系结构，只要还有**RAM**可用，栈空间就能自动增长。不过即便如此，算法的某些执行路径所需使用的栈空间可能还是会比全部可用的**RAM**还多。

减小栈溢出发生几率的一种办法是尽可能地减少全局变量的数量。全局变量很有吸引力，特别是对于初涉编程的人，因为它们在哪里都是可用的。不必操心如何在函数之间传递它们，也无需搞清为什么编译器会抱怨某个变量没有被定义，尽管在你看来，它显然就在那里.....在另一个函数中.....然而，因为总是需要为它们分配内存（如此前补充资料中的示例所示），任何全局变量都会一直占用宝贵的**RAM**空间。相比之下，局部变量只存在于它们的声明所在的函数中（假设你没有使用**static** 之类的修饰符使它们持久存在），因此只有在被需要的时候才占用空间。如果能把更多的变量移到实际会运用到它们的函数中，就能释放更多空间，以供执行路径上的其他部分使用。

另一个限制栈的使用量，或者至少让栈的使用量小于明确定义的边界值的办法是，避免使用递归算法。尽管优雅的递归算法能让代码更易理解，但每次递归都会导致栈的增长。对于预期的输入，如果了解某个给定函数会递归多少次，这可能不是一个问题。但如果你不能确定这个算法是否会递归很多次以至于让栈溢出，那么，对于嵌入式系统，最好以迭代的方式重新实现这个算法。迭代算法具有已知的栈使用量，而递归算法的每一层递归都会增加栈的占用量。

8.2 性能和电池寿命

谈到编写代码，性能和电池寿命往往如影随形——有利于其中之一的举措通常也对另一个有好处。需要对二者之一还是同时对二者做优化，取决于应用本身。例如，被栓在固定位置、由插在墙上的电源适配器供电的装置对节能没有太大的需求。不过，低耗能应该是所有装置追求的特征。

同样，如果正在构建的物品不需要即刻做出反应（它可能是一个融入周边环境的天气预报通知装置，晚更新几秒钟不会产生任何不良影响），或者不具备需要快速响应用户操作的交互式用户接口，那么就不太需要最大限度地提升性能。

对于用普通电池或太阳能电池供电的物品，和那些需要在用户按下按键时瞬间做出反应的物品，对性能或功耗给予适当的关注是非常有意义的。尽管高德纳所说的“过早优化是万恶之源”（或者是Edsger Dijkstra说过这话，对此没有统一的说法，<http://hans.gerwitz.com/2004/08/12/premature-optimization-is-the-root-of-all-evil.html>）很有道理，但了解一些适用的技术还是有用的，将有助于在需要做抉择时选择更有效率的算法作为默认之选，而非随意去选。（不过，我们坚信，如果效率问题还不明确，至少在对代码进行分析并找出需要优化的地方之前，代码的可理解性、可维护性要比效率重要。）

很多重要的功耗控制收益源自于硬件设计，特别是如果当系统中的模块不在使用状态时，装置要能关闭它们，或者当代码的执行过程结束或正等待某个事件发生时，整个处理器能进入低功耗休眠模式，那么你就实现了基本的节能降耗目标。尽管如此，对软件也做一些优化仍然很重要——毕竟，越及早执行完主要代码，硬件就能越早进入休眠状态。

为了使代码更高效，一个最简单的办法是使用事件驱动模型，而不是通过轮询检测变化。这样做能让装置有更长的时间处于低功耗状态，并且只在需要时才快速切换回正常操作状态，而根本不用为检查是否有变化和待做任务而周期性地做无用功。如果装置是客户端，而不是

作为一个等待消息到来的服务器，那么在建立模型时，创建联网相关代码就会比较复杂。诸如第7章介绍的长轮询，类似于消息队列遥测传输（MQTT）或基本的Socket连接的协议，这些技术都能解决这种情况下的联网问题。

硬件方面，考虑使用处理器特性（如比较器或硬件中断）来唤醒处理器，并且仅当相关的传感器满足条件时才调用处理代码。如果代码需要在给定时间段内暂停执行，以便在继续运行之前允许某种效果出现，那么就调用适当的函数令处理器进入休眠状态，而不要在忙循环中等待。

减少正在处理的数据数量也有一定的助益。你正在使用的某个服务的API可能已提供了选项，能降低发送给你的信息量。例如，当从某个Twitter账户下载推文时，只要求得到指定ID后的推文。在第一次调用这个API时，就不得不处理它发送来的所有推文。但在随后的调用中，就可以只要求得到已处理过的最新推文ID后的推文。

这种优化只有当正在使用的API提供了此类特性时才能实现。对于很多操作，限制其行为没有任何意义。如果对数据作限制有意义，但现有的API没有提供这种可能性，则可以选择写一个自己的服务（被称为**shim服务**），使其位于装置和正确的API之间。Web服务器拥有的处理能力和存储能力，shim服务也都具备，因此它完全可以承担大多数繁重的工作。这样做之后，shim服务只会发送最少量的数据给嵌入式系统做处理。

例如，Bubblino直接与Twitter通信，每次检查新消息时，都会下载XML格式的完整搜索结果。所有这些数据经过处理，最终都会缩减为单一的数字，即发现的新推文数量。理论上说，经过优化的中介服务可以执行搜索并只给Bubblino传送一个数字。但这样做的缺点在于，需要编写另一个服务，并维护支持其运作的基础设施。此外，随着产品销量增长，越来越多发送到因特网上的请求会被汇集到中介服务，经由“Bubblino服务器”和推特服务器之间的路径传送。在这种情况下，无论Bubblino装置本身在哪里，最好还是让这些请求流量分布到全球的各个地方。

谈到算法本身的原始性能，没有什么能胜过性能分析。它能帮你搞清楚速度的瓶颈在哪。然而，如果你能牢记下列几个习惯，你的代码总体上将更有效率。

- 编写**if/else** 结构，在两个可能的执行路径间做选择时，尽量把更可能被执行的代码放到第一个分支，即**if** 部分，而不是**else** 部分，如下所示：

```
if something is true
    The more likely to happen code goes here
else
    The less likely path of execution should go here
```

这使得在较常见的情况下，指令的预取和使用先行控制技术的指令流水线将发挥作用，不会让预取的指令被丢弃并重新填充流水线。虽然让流水线重新做好准备只需几个额外的处理器周期，但小的改进也是有帮助的。

- 在“内存管理”一节，可以看到，如果已知某个数据永远不会变化，可以把它声明为常量，这能帮助编译器将其放入闪存或**ROM**，也有助于让编译器知道怎样优化代码。在某些情况下，在代码中插入一个数值时，使用普通的数字要比从内存中某处一个变量所在的位置加载要快。如果编译器知道变量的取值将始终是什么值，它就能做这个替换。
- 避免到处复制内存中的数据。在内存中移动大块数据到各处是真正的性能杀手。在理想情况下，代码应该只查看它所需要的数据，并且只读取一次。在实践中，实现这一结果非常困难，但这不失为一个好办法，能有助于修正你关于如何编写代码的看法。在处理协议时，这尤其是一个问题。数据先是被读入一个数据缓冲区（如以太网层的缓冲区），然后被依次传递给**IP**层和**TCP**层，接着是**HTTP**代码，最终到达应用层。一种简单的方法是，在分析相关的协议头时，每一步都复制数据。这种方法会导致所有的应用数据在协议栈中向上传递时被复制5次。更好的办法是在各

层之间传递一个指向初始缓冲区的指针或该缓冲区的一个引用。你除了需要保存数据的长度，可能还要保存一个缓冲区中的偏移量，以指明相关数据的起始位置。这样做会使得复杂度稍有增加，但能大幅度减少需要复制的数据量。

- 和前一点相关，当确实需要复制数据到各处时，系统提供的内存复制和移动例程（如**memcpy** 和**memmove**）通常做得更好，能更有效率地复制数据，所以请使用它们。在可能的情况下，特别是在32位处理器（如ARM系列）上，它们会使用处理器指令，单次操作就能复制多字节，从而大大加快处理速度。

8.3 库

现如今，在为服务器或台式机开发软件时，人们通常可以轻易获得大量可能用得上的库和框架。它们可以让生活更轻松。需要解析大量的XML格式的RSS数据？没问题！只需把所选用语言的RSS解析库找出来即可。想发送电子邮件？不用担心，有现成的模块专做这件事。还可以举出很多这样的例子。

在嵌入式的世界中，做这些事情往往会有点棘手。随着片上系统产品的兴起和嵌入式Linux在这些产品中的使用，情况正逐渐好转。就像在“正常的”Linux系统上操作一样，大多数的服务器软件包能够以同样的方式被添加到嵌入式Linux系统中。最棘手的部分可能是：如果某个库没有针对你的系统（如ARM）的一个现成已经预先构建好的版本，那么要设法搞明白怎样重新编译这个库，使其适合你的目标处理器。

另一方面，微控制器在资源方面仍然非常受限，主流操作系统上的库和代码不是拿来就能用的。你也许能在这些代码的基础上编写自己的版本，但如果这些代码做了大量的内存分配或处理工作，可能最好还是从头开始编写，或者找一个在编写时就已经考虑过微控制器的各种限制的版本。

在这里没有足够的篇幅对所有能获取到的可能用得上的库都做介绍，并且我们也绝对不是对任何可用的库都有所了解。不过，你可能会对下列几个库感兴趣。

- **lwIP**：lwIP，即轻量级IP
(<http://savannah.nongnu.org/projects/lwip/>)，是一个能在低资源条件下运行的完整TCP/IP协议栈。它只需要几万字节的RAM和大约40KB的ROM或闪存。Arduino官方授权的WiFi盾板使用了这个库的一个版本。
- **uIP**：uIP，或micro IP
(http://en.wikipedia.org/wiki/UIP_%28micro_IP%29)，是一个以可实现的最小系统为目标的TCP/IP协议栈。它甚至能在只有几千字节RAM的系统上运行。它是通过不使用任何的缓冲区对传入的

分组或发送出去的未经确认的分组进行存储实现这一点的。这意味着TCP层的一些重传逻辑需要由应用程序实现，这使得代码的耦合度增加，变得更复杂。在没有使用标准的以太网盾板和相关库的Arduino系统上，如Nanode板，使用Ethercard在AVR上的移植版本是非常常见的（<https://github.com/jcw/ethercard>）。

- **uClibc**：uClibc（<http://www.uclibc.org/>）是一个面向嵌入式Linux系统的GNU C语言标准库（glibc）。它需要的资源比glibc少得多，几乎可以直接替换掉glibc。把使用的C语言库变更为uClibc时，通常只涉及源代码的重新编译。
- **Atomthreads**：Atomthreads（<http://atomthreads.com/>）是一个用于嵌入式系统的轻量级实时调度器。当代码变得足够复杂，以至于需要在同一时间做几件事情时（不是真的同时，但调度器在任务间切换的速度足够快，以至于看上去像是同时发生的。这和PC机上的多任务是同样的道理），可以使用它。
- **BusyBox**：尽管不是一个真正的库，BusyBox（<http://www.busybox.net/>）将大量有用的UNIX工具集成到一个单一的可执行文件。它是一个常见的、有用的软件包，能在你的系统上提供一个简单的shell环境和各种命令。

8.4 调试

编写软件时，最令人沮丧的一件事情是，知道代码有错误，但却完全不清楚错在何处。在嵌入式系统中，由于检查正在发生的事并追查问题所在位置的手段往往较少，这种情况会让人觉得倍加沮丧。

由于引入了定制的电子电路（它们可能会行为异常或设计不正确）以及经由网络与服务器的通信，构建物联网装置会让事情进一步复杂化。排除电子电路故障不在本书的论述范围之内，但我们将介绍一些对网络通信进行调试的方法。

现代的软件集成开发环境（常被缩写为**IDE**），对深入挖掘代码运行时正在发生的事情有着很好的支持。你可以设置断点。当一组预定义的条件被满足时，它们就能让代码停止执行。此时，你可以在内存里到处看看，看里面有什么内容；可以对表达式求值，看假设是否正确；然后一行一行地单步执行代码，看会发生什么。你甚至能在调试过程中修改内存的内容或变量的取值，从而对后续代码的执行施加影响。在更先进的系统中，甚至能在程序停止运行时改写代码。

嵌入式系统的调试环境通常比较原始。如果你用的嵌入式平台能运行功能较为完备的操作系统，如**Linux**，那么，相对于在很小的微控制器上做开发，你处于比较有利的位置。

像嵌入式**Linux**这样的系统，通常会支持使用**gdb**（GNU调试器，www.gnu.org/software/gdb/）之类的工具进行远程调试。此类工具允许你通过某种连接方式，把桌面系统的调试器附着到嵌入式单板。连接方式通常是采用串行连接，但有时也会使用以太网或类似的网络链路。一旦附着成功，你就可以使用一系列类似于在台式机上调试的功能，如设置断点，单步执行代码，检视变量和内存中的内容等。

另一种获得桌面级调试工具的方法是在台式机上模拟目标平台。因为你是台式机上运行代码，和调试桌面应用程序相比，你可以使用的功能是一样的。这种方法的缺点是，你不是在工作现场的真实硬件环境下运行程序。

尽管这种方法非常有用，能用来发现一些初期的错误，让程序大体上运转起来，但可能存在这种情况，即对于某个奇怪的问题，你无法用这种方法捕获到，只有当程序在最终的硬件上运行时，它才能暴露出来。

如果软件特别复杂，模拟是个好的调试手段，因为在这种情况下，你需要花很多的时间进行开发和调试。然而，目标硬件与台式机之间的差别越大——特别是在涉及具备专用传感器或执行器的硬件时，为模拟器编写能精确反映电子器件行为的软件子系统的难度就越大。

如果你需要在目标硬件上调试，并且平台不允许你使用gdb（或者串口被系统的其他部分占用），JTAG访问可以为你提供需要的功能。JTAG是以提出该标准的行业组织（the Joint Test Action Group，联合测试工作组）的名称来命名的。制定JTAG的最初目的，是为已经安装好元器件的电路板提供一种测试的手段，而在当前，这仍然是一项重要的用途。

然而，从JTAG出现开始，它就在功能上得到扩展，提供了更高级的调试功能。从软件角度看，最让人感兴趣的是在与单独的PC机上的某些软件连接后，JTAG具有的一些被称为在线仿真器（ICE）的功能。它们允许你使用其他计算机来设置断点，对运行在目标处理器上的代码进行单步调试，并且大多还能访问寄存器和RAM。有些系统甚至允许你利用复杂的硬件事件触发调试器。和使用gdb之类的调试器相比，你能获得更好的控制或访问能力。

如果你无法使用上述的任何工具，你将不得不转而依靠一些更简单的，但已经过实践检验的技术。

最显而易见和最常用的穷人的调试技术是把字符串输出到一个日志系统。几乎所有的软件都会用到这个方法，它使你能够把任何你认为有用的信息写入日志。这些信息可以是某些变量在代码中的关键点的取值。或者，如果你怀疑系统的RAM会逐渐耗尽，可以在程序启动时，以及程序启动后在代码中不同的位置，把可用内存的数量写入日志，这将有助于搞清是否存在这个问题。如果代码在执行过程中看似进入挂起状态，诸如“运行到第X行”这样的一些简单的调试输出能让你利用二分法缩小查找问题所在的范围。

如果你能访问一个可写的文件系统，如SD卡上的文件系统，你可以把输出信息写入一个文件，但更常见的做法是把信息写入一个串口。这种方法使你能在串行连接的另一端附着一个串口监视器程序，如Windows上的超级终端程序，从而能（几乎）实时看到输出信息。

不过，你应该意识到有两个意料之外的问题存在，尽管在很多情况下你是碰不到的。首先，记录日志显然是需要耗费一定量的空间的，所有的用来记录日志的字符串和代码都必须与其他代码一起装载到嵌入式系统中。

刚才提到能“几乎”实时看到调试信息，“几乎”这个措辞暗示了第二个问题的所在。因为串行通信要恪守规定的速率，通常在发送数据时，会使用一个小的缓冲区存储正在发送的数据。如果你的代码在执行输出日志信息的操作后，很快就挂起或崩溃了，那么就有可能在系统停止运作之前，来不及通过串行连接把日志信息发送出去。因此，如果你想通过打印“到达此处”之类的信息，搞清楚程序在遇到错误之前能执行到哪里，实际到达的位置有可能比串口输出的日志中提示的位置更远一些。不仅仅是在通过串口输出日志时才有此问题；文件系统往往会设置缓冲区，用来把数据写入文件，因此使用日志文件可能也会有相同的问题。

由于大多数物联网装置能持久地保持联网状态，可以在网络接口功能中添加一个调试服务。这样一来，就可以创建一个简单的服务，从而能够使用telnet之类的基本工具连接该服务，并实时地对正在发生的事情做更多了解。在最简单的情况下，该服务可以输出日志数据，要不然的话，这些日志只能定向输出到串口。或者，可以让该服务理解若干编程者自定义的简单命令，从而用这些命令查询系统各个部分的状态、触发功能或测试代码。使用这种方法时要格外留心，不要无意中开启一个后门安全漏洞。这是一个显然需要警戒的地方。我们不建议在最终的产品中保留这项调试服务。

其实，即便没有专用的网络调试接口，也可以利用装置具有网络连接这一事实，设法弄清楚什么地方出错了。

即便上层的代码不能做你期望做的事情，TCP/IP协议栈通常也能提供基本的调试功能，如对ping命令的ICMP请求做出响应。如果知道装置

的IP地址，并且当使用网络工具ping对其进行查询时，该装置能做出响应，你可以推断出，至少系统的某个部分还在运作。

更进一步的做法是，在装置和与之通信的服务之间的网络路径上，如果你能在该网络的某处连接一台计算机并在其上运行一个数据包（分组）嗅探器的话，你就能看到网络层面正在发生的事情。

（Wireshark, <http://www.wireshark.org/>，是作者们在这种情况下的通常选择。）通常，你是在网络连接的某一端实施监测，或者把计算机连接到装置所在的局域网中，或者在远端的服务器上运行数据包嗅探器软件。

除非网络流量很小，否则你需要使用过滤选项，把信息量减少到可管理的程度。如果你是在装置一侧进行嗅探，根据装置的MAC地址进行过滤是一个好办法；否则，可以使用装置的IP地址，把显示内容限制为出入装置的流量。这种做法使你能够看到装置是否在进行网络注册（如果有DHCP流量），是否成功地连接到了远程服务，并能发送出相关的数据，得到正确的响应。我们希望所有这些都能有助于你缩小查找问题的范围。

在更高一些的层面上，你也可以使用服务器上的日志记录功能和软件，对装置的活动信息进行收集。如果装置和服务之间的传输采用的是标准协议，例如，使用HTTP与Web服务器通信，那么，很有可能任何请求都会被记录到服务器的日志文件中。如果还能修改服务器侧的API，可以为其增加分析功能，例如，对客户端上一次获取服务的时间，或访问API的次数，进行跟踪记录。

如果上述调试方法都行不通，还有最后一招。你在构建硬件时的第一个步骤，可能是让LED灯闪亮，改变其闪烁方式算是（通常未必是）最后一种调试手段。只要有一个空闲的GPIO引脚，就应该能够连接一个LED灯，并让代码在指定位置点亮这个灯。和使用字符串记录日志的方法一样，如果系统挂起，可以使用这个技术缩小查找问题的范围，或者，用不同的闪烁模式来表示不同的条件也是可能的。

对于在系统的其他地方正常使用的LED灯，你也能够对其重复利用。至少在调试阶段可以这样做，只要不同的用途可以被相互区分开来。记得艾德里安在调试一个手机Web浏览器的网络协议栈时，用到了这

种方法。指示底层活动的唯一途径是让双色（幸好是双色）**LED**状态灯闪烁，用一种颜色代表传入的数据包，用另一种颜色代表发送出去的数据包。

8.5 小结

虽然本章提供的用于改进嵌入式编程的方法，绝对不是全部，但我们希望本章的内容能提供一些有用的提示和建议。为了便于参考和唤醒你的记忆，下面再重温一下要点。

- 把尽可能多的数据等各种内容移到闪存或ROM中，而不是放在RAM中，因为后者往往不太够用。
- 如果某些数据项不会改变，把它们定义为常量。这让它们更容易被移入闪存或ROM，也让编译器能更好地优化代码。
- 如果内存很少，在堆和栈之间要优先选择使用栈。
- 谨慎选择算法。单次扫描算法能够处理比全部可用内存多得多的数据。选择使用迭代而不是递归能使内存的使用更有确定性。
- 为了使电量使用最佳化，应该让系统尽可能久地保持在休眠状态。
- 如果不是正在使用（无论正在使用什么），那么就要尽可能关闭。此建议不仅适用于处理器（转入低功耗模式），也适用于其他硬件子系统。
- 除了装置本身，服务器侧也需要优化。采用非轮询的方式，或者减少传输的数据量，对解决方案的两端都是改进。
- 避免过早优化。如果碰到了性能问题，可以通过分析代码搞清楚问题所在。
- 复制内存数据的代价高昂，尽量少做此类操作。
- 与编译器协作，而不要与之对抗。让代码帮助编译器确定可能的执行路径，并且通过使用常量帮助编译器做优化。

- 慎重选择库。对于更为紧凑的嵌入式环境，来源于标准操作系统的库可能不是好的选择。
- 在调试时，诸如gdb和JTAG之类的工具是有用的。不过，只凭借输出文本到串行终端，或让一个LED灯闪亮，也能完成很多调试工作。
- 仔细观察装置周边的环境，能帮助发现问题，特别是当它连接到了因特网，能与更广阔的世界进行互动时。

第二部分 产品阶段

- 第 9 章 商业模式
- 第 10 章 生产制造阶段
- 第 11 章 道德伦理

第9章 商业模式

如果你的主要身份是一名创客或程序员，而不是企业家，你对“商业模式”可能只有模糊的概念。在非正式的讨论中，这个表述差不多专指企业如何盈利。例如，当人们谈论Twitter、Pinterest或其他影响广泛的新社交媒体时，通常会这样贬损它们：“它们找到自己的商业模式了？的确，它们现在做得很大，但在如何盈利方面有什么想法吗？”

但商业模式不仅仅只和金钱相关，它还有更多的内涵。我们可以把商业模式定义为一种假说，这种假说更多地与以下状况有关：客户需要什么，希望它是怎样的，以及一个企业该如何组织以最好地满足这些需求，这样做之后如何获得报酬并从中获利。

(www.sciencedirect.com/science/article/pii/S002463010900051X)

这个定义汇集了若干要素：

- 一群人（客户）；
- 这些客户的需求；
- 企业可以做的用来满足这些需求的事情；
- 有助于实现这一目标并能持续地继续这样做的组织行为；
- 一个成功的标准，譬如盈利。

所有这些方面都与业余爱好者或非盈利项目相关，其相关程度并不亚于商业企业。不过对于上面最后一个要素，可能需要把盈利替换为“让世界变得更好”或“获得乐趣”，以此作为成功的标准。

我们在这一章中首先将概述商业模式的发展历程，对这个话题有所了解，然后将介绍一种能演化发展成商业模式的常用方法。随后再来看看现有的物联网公司是如何构建他们各自的商业模式的，并探讨一下他们将来的发展情况。最后，我们将从初始融资开始，来实际了解一下怎样创办公司，并对“精益创业”方法的优点进行讨论。

9.1 商业模式简史

远古以来，自人类出现后的大多数时间里，我们是以部落的形式聚居，共同拥有财产，共享资源的。原始狩猎采集者普遍采用这一模式。部落中的每位成员都能够获得食物和住所，即使他们某天运气不好，没有找到食物或打到猎物。我们把这种形式的集体主义看作是一种基本的**礼物经济**。具有特定技能的人提供他们的产品或服务，如陶器、牲畜、粮食、代为狩猎、照看小孩等，并不期望立即获得与这些产品或服务对应的报偿，而是在以后获得价值相当的礼物，这样就形成了礼物经济。这不是一笔书面债务，而是一种社会责任。受惠者将在适当的时候做出偿付，也许是当狩猎收获多时，也许是她碰巧找到她的手艺所需的原材料时，或者甚至是在当年很晚时的收获季到来时。

诸如易货和货币之类的系统，只是在不同部落之间的边缘地带发展起来的。我们可以认为，最早的能被我们看作是现代商业模式的系统是在部落边界发展形成的，并且源自于在空间和时间上移动产品和服务所需的技术。

9.1.1 空间和时间

相邻的几个部落可能会发现，他们各自拥有的本地资源，如动物、蔬菜或矿物，会有所差异，但只有当 they 与遥远地区发展贸易时，这种差异才会变得真正有趣。商人可以把在他所在村庄制作的丝绸卖到这类布料稀缺而又对此有需求的地方，并以此换取在其家乡价格昂贵的香料。但长途贸易带来了一系列的问题：虽然游牧式的狩猎采集者善于寻找食物，以及在迁徙途中安居，但商人们必须携带大量的待售货物，而且不希望在生活和居住方面浪费太多时间，以便能够将大多数时间用于运输。他们携带的货物和食物必须能长久保存，所以需要保护和贮藏。综上所述，需要有一种可靠的运输方式，用来运送商人及其商品。技术的进步使得人们可以通过水路以及水陆联合运输等多种途径来运输货物，从而开辟出新的路线。同时，由于人们驯化了骆驼等动物，最终开辟出穿越阿拉伯西部沙漠的贸易路线。

我们已经简略地提到了食物的贮藏，它是通过盐腌、烟熏，或者只是采用更好的储存技术，如谷仓，来实现的。同跨越空间运输货物一样，贮藏是一种跨越时间运输货物的方式。农民或商人，如果能在收获季节和农产品过剩时不吃掉或卖掉所有的农产品，就可以在几个月后以更高的价格将产品出售。因此，商人的业务就是跨越空间和时间运输货物，而他们的供应商和生产者则能够通过一次性地卖掉大量农产品而受益。此后，他们就可以继续日常生活和工作了。

接下来，货币通过在固定体量的通货（一定尺寸的金饼或一定重量的稻谷）和被交换的产品之间设定一个易于计算的交换率，进一步抽象了贸易过程。在原始的礼物经济时代，生产者只能根据狩猎、耕作或手工制作的节奏，周期或间歇性地偿付商品和服务。有了货币之后，这些商品和服务就被抽象出来，能在任意时间被偿付。从这个意义上说，货币是另一种跨越时间的技术。货币发展带来的通用性和易于计算的好处，使得发展新的商业模式变得更加容易，如投资其他商人的长途贸易以换取一定的股份，或者发放贷款以获取利息等。

9.1.2 从手工制作到批量生产

大约在公元1450年前后，古腾堡发明了印刷机，从那之后，书籍就从由僧人和工匠手工制作的稀世珍品，转变为可以被生产的商品。很快，每个资产阶级家庭都能买得起属于他们自己的图书，至少是一本古腾堡圣经，这是第一本能够批量生产的书籍。毫不夸张地说，该项发明奠定了当前以因特网和万维网为典范的信息文化的基础。不辞辛苦地把古代文字复制到羊皮纸上，与采用最新的技术创新，把这些文字冲压到纸上，看似产生相同的结果——一本书，但后者要快几千倍。这不是一个简单的量变（生产出更多的书籍），而是一个质变：信息不再稀有、宝贵和脆弱，不再需要守护者（统治阶级和教会）看管，能够被广泛传播，以至于每个人都可以有机会获取（无论那些之前的守护者们愿不愿意，这终将会成为现实）。

在15世纪末，印刷机经由海路传播到新大陆和印度，贸易航线在其中也发挥了很大作用。随着技术的传播，印刷成本越来越低，报纸和小册子开始兴起，从而产生了新的商业模式。到了19世纪中叶，狄更斯写小说时，他已能够采用按月或按周订阅的方式，一次发表一章内容了。

1884年，英国利华兄弟公司推出了“阳光”牌肥皂。这是第一种包装成块并且带有品牌标识的家用肥皂。在这之前，肥皂是由杂货店的人们切割，并按重量售卖的。这是一种在大众消费领域的创新，凭借品牌直接在消费者和生产者之间建立了信任关系，从而降低了中间商和杂货店的地位，让他们成为只是把产品交付给消费者的一种途径。

批量生产是商业模式的另一重大变革。福特汽车公司完善了这一模式，但其驱动力不是来源于亨利·福特如何售车，而是他怎样造车。福特放弃了“手工生产”模式，即按照高度自定义的需求接受委托，再由熟练的工匠制作出来。而是让他的工人们专注于单一的任务，他坚持采用标准规格的零部件，使得汽车可以被装配起来，最终形成相同的产品。这种做法使得福特汽车变得易于维修，也让普通人也能消费的起，无需雇佣机械师来维持其运行。批量生产带动汽车生产成本下降的事实，也有助于让汽车保持在一个大众能够承受的价位。

商业模式转变到批量生产有其自身成本，尤其是半熟练的工厂劳动力与被其替代的技艺多样化的工匠相比，其技术水平可能不太令人满意。再考虑上社会成本，典型的批量生产的运作在效率方面已遇到瓶颈。丰田公司在20世纪50年代开创的精益生产方法，在保留了批量生产的很多要素（效率、自动化和大批量生产）的同时，还能根据订单要求在指定的时间生产产品，而不是单纯批量生产单一的零件、组件和成品。因此，丰田公司可以单独生产符合给定颜色和车轮配置等要求的汽车，并且能够在要求的时间和地点，让正确的轮毂、轮胎和特定样式的车门到达生产线上。和批量生产相比，这种方式允许该公司实现更大程度地定制化生产。而强调对效率的持续改进，被认为可以为工厂中的工人创造一个更有意义、更多样化的环境。

在其他领域，批量生产的观点促成新的商业模式的出现，如超市。超市率先实现了自助购物和在同一屋檐下销售一整套的产品。最早的超市被认为出现在20世纪30年代，并在60年代演变为大卖场。现在，自助购物的概念已经发展到了自动收银，每位购物者都能成为他们自己的结帐助手。

快餐特许经营开始于20世纪30年代，并且随着麦当劳和汉堡王的出现，在50年代出现了爆炸式增长。标准化的菜单，事先准备好的配料，每个加盟店都要遵循的标准操作规程，意味着你现在可以在本国

之内的任意一家连锁餐厅吃到一模一样的食品（在全球范围内，快餐店会根据当地的口味、法律和宗教，对菜单略微调整）。一个有趣的变化是，很多新的快餐连锁店正在对抗这种做法。它们添加的酱汁是采用新鲜食材手工制作的，而不是批量生产的。总部设在美国的Chipotle连锁餐厅是*Fast Company*杂志评选出的2012年度最具创新的50家公司之一。在成功的快餐业商业模式基础上，该连锁餐厅的采购过程更加合乎道德规范，制备食品的员工也承担着更多的责任，这些特点使餐厅重新获得新生。与其类似，Lush公司在20世纪90年代也创建起一个香皂帝国，其做法是销售尚未按重量切割成块的长条天然皂，最终由顾客按照自己的喜好来切割称重，采用的正是一个世纪前利华兄弟公司进入该领域前业界的通常作法。

9.1.3 因特网时代的长尾效应

正如我们所看到的，技术变革通常能促成商业实践的巨大改变，或者，技术变革的结果会带来商业实践的巨大变化。20世纪最大的一个技术模式转变就是因特网。从蒂姆·伯纳斯·李在20世纪90年代首次向人们展示万维网开始，到eBay和亚马逊开张，只用了5年时间。又用了5年时间，它们不仅成为了互联网泡沫的幸存者，也是胜利者。它们改变了我们买卖物品的方式。《连线》（*Wired*）杂志的克里斯·安德森提出并推广了“长尾理论”这个术语，并用它来解释这个巨大转变背后的机理。

实体店需要支付房租和维持库存。所有的库存都需要占用宝贵的店面空间，因此，实体店会着重提供经常光顾的顾客们会买的东西：最流行的“热门”商品（或“短头”）。相比之下，因特网的店面展示的只是比特信息，它实际上是免费的。当然，亚马逊也需要维护仓库和库存，但和面向公众的实体店面相比，管理起来更加有效率。因此，亚马逊可以把数量众多的商品推向市场。虽然其中的一些产品可能不是特别畅销，但如果把所有这些产品的销售量都加起来，数目仍然十分巨大。

在利物浦、美国俄勒冈州的斯普林菲尔德，或意大利的佛罗伦萨，一家专营商店可能会（也可能不会）找到足够多的顾客来维持其商业定位——这取决于城镇的规模和文化的多样性；而在因特网上，所有的商业定位都能找到市场。实践长尾理论的因特网巨头们通过把更小的

供应商提供的产品聚合起来，正如亚马逊的集市，或eBay上的卖家所实现的那样促进了这一进程。这种方式不仅促成了成千上万家小规模
的第三方贸易商的存在，也令提供聚合服务的商家赚到了钱，并且完全不用操心库存或配送的事情，因为这些事情已经外包给“长尾”了。

电子书和按需印刷也在改变出版界的面貌。它们可以使用更为广泛各种可用材料，让作家和出版商之间如今仍旧存在的传统商业模式产生连锁变化。随着谷歌颠覆了存在不到十年的搜索引擎世界，新的商业模式不断被创立，又不断被颠覆。然而，尽管谷歌宣称它的目标是“整合全球信息，使人人皆可访问并从中受益”（www.google.com/about/company/），但它主要是还是通过利用广告长尾，允许小生产商和大公司一样有效地投放广告，进而从中获利。

9.1.4 以史为鉴

通过对人类历史的快速的回顾，我们已经看到了商业模式的若干要素。但我们学到了什么可以应用到物联网项目中的内容，能把项目变成一个可行的盈利业务呢？

首先，我们看到一些模式历史悠久，例如，制造销售物品。虽然制造或售卖物品的方式可能会改变，但这个基本原则已沿用千年。

其次，我们看到了新技术是如何激发出新商业模式的。我们尚未尽述因特网和万维网促成的所有新商业类型。如果我们相信物联网可以代表一种类似的技术巨变，则伴随而来的将是我们今天很难想象的新商业模式。

第三，尽管有重复出现的模式和常见的模式，但它们也有无数的变种。对某个因素，如制造过程，或者产品或资源的支付方式，做细微的改变，能对整个业务链造成连锁反应。

最后，新商业模式具有改变世界的力量，就像品牌肥皂迎来了大众消费，以及批量生产改变了工作本身的概念一样。如果物联网的确能改变世界，那它很可能是通过采用允许的商业模式来实现的，正如我们接下来将在第11章中讨论的那样。

9.2 商业模式画布

一个最流行的商业模式设计模版是由Alexander Osterwalder和他的初创公司Business Model Foundry提出的商业模式画布。该画布是一个采用知识共享许可的单页规划图，如图9-1所示。

商业模式画布

设计用途

设计者:

日:

月:

年:

迭代编号

<div>关键合作伙伴</div> <div>谁是我们的关键合作伙伴? 谁是我们的关键供应商? 我们正从合作伙伴处获取哪些关键资源? 合作伙伴们从事哪些关键活动?</div>	<div>关键活动</div> <div>我们的价值主张需要什么关键活动? 我们的分销渠道? 客户关系? 收益流?</div> <div>关键资源</div> <div>价值主张需要什么关键资源? 分销渠道? 客户关系? 收益流?</div>	<div>价值主张</div> <div>我们给客户传递什么价值? 我们要帮助客户解决哪个问题? 我们正在为每个客户群提供什么成套的产品和服务? 我们正在满足哪些客户需求?</div>	<div>客户关系</div> <div>每个客户群希望我们与之建立和保持何种类型的关系? 已经建立了哪些关系? 如何把它们与商业模式的其余部分进行整合? 成本如何?</div> <div>渠道</div> <div>客户群希望我们通过哪些渠道与之接触? 目前使用的渠道? 这些渠道如何整合? 哪些渠道更有效? 哪些渠道成本效益最好? 如何将渠道与日常客户工作整合?</div>	<div>客户群</div> <div>我们为谁创造价值? 谁是我们最重要的客户?</div>
<div>成本结构</div> <div>我们的商业模式中最重要的固有成本是什么? 哪些关键资源花费最多? 哪些关键活动花费最多?</div>		<div>收益流</div> <div>让客户真正愿意付费的是什么价值? 他们现在付费买什么? 他们目前如何支付费用? 他们更愿意如何支付? 收益流各分支对总收益的贡献率是多少?</div>		

图 9-1 商业模式画布

依据知识共享许可协议3.0版，转载自businessmodelgeneration.com。

乍看这张图，会觉得其中的每个方框就是一个要素，整个图可以用一个包含九个要点的清单代替。然而，方框被设计为适当的尺寸，能正好用来容纳即时贴，从而突出了可以把玩的各种现有想法，并且能到处移动它们。此外，图的布局赋予了每个项目一定的含义和语境。

让我们看一下这个模板，从最明显的要素开始，然后深入到更为逼真的细节。如果没有这种模板，我们可能会忽略这些细节。

右下角是收益流，它或多或少的对应了“如何盈利”这个本章开篇的问题。尽管它的位置表明，它的确是企业最重要的期望成果之一，但它绝不是唯一的考虑因素。

中间的方框里是价值主张。其更直白的说法是，你将生产什么——其实就是你的物联网产品、服务或平台。

客户群是你计划向其交付产品的对象。它们可能是其他创客和极客（如果正在生产工具包形式的装置）、普通大众、家庭、企业或43岁的会计师（哈雷戴维森公司假想的著名一般客户）。

客户关系可能涉及公司和其最热情的客户间经由社交媒体进行的持久通信。这种情形能够传递一种优势，但可能维护起来代价昂贵。维护一个顾客“社区”可能有益，但你将优先处理哪些关系，以便与最有价值的客户群保持沟通呢？

渠道是接触客户群的方式。从做广告、分销产品，到配送和售后服务，你选择的渠道必须是和你的客户相关的。

模板的左侧是几个不能或缺的要素。没有它们，就没有产品可供出售。关键活动是指需要去做的事情，如制造物品、编写代码。也许还包括一个运行它的平台、一个网站和实体产品的设计。

关键资源不仅包括需要用来创建产品的原材料，还包括帮助构建产品的人。你拥有的智力资源（数据，以此类推，还包括专利和版权）同支付这一切所需的财力一样，也是有价值的。

当然，很少有企业能有足够的财力和时间，独自完成所有的关键活动，或者去调度所有的关键资源。（亨利·福特曾努力尝试这样做，但即便是他也没有做到。）你需要关键合作伙伴，即能够更好地提供专业技能或资源的企业，因为那是他们的商业模式，和你自己做相比，他们已准备妥当，能以更低成本做某件事，或者能把事情做得更好。也许你委托一家组织进行Web设计，并使用全球化的物流公司做运

输工作。你会自己制造一切，还是让供应商制造组件，甚至组装整个产品呢？

成本结构要素要求你给刚刚定义的资源 and 活动标上价格。它们中哪些最为昂贵？给出你将承担的成本后，这种分析有助于你决定是更多地采用成本驱动策略（采用低价出售，销量大，通过自动化和提升效率增加产量）还是价值驱动策略（以更高的利润率销售优质产品，但销量相对较少）。

9.3 商业模式的用途

为你的企业建模的首要原因，是为了提供某种基于经验的假说，即你的企业是否能传递你所期望的事物。即使你不使用我们刚讨论过的画布这样的半形式化方法，任何人在创立任何企业时，都会至少简单地考虑一下，他是否承担得起，要做的业务是什么，以及能否盈利。

作为一名程序员或创客，你可能会认为，把一张画有九个方框的纸看作“工具”是有悖常理的。但是，当你能采用行之有效的方式把要考虑的因素分割开来时，画布提供的少量结构能帮助你思考业务，并能让你思考不同想法来一场头脑风暴：

- 如果我们的产品目标用户是学生，而不是企业，将会怎样？
- 如果我们把设计外包给某个机构，将会怎样？
- 如果我们采用数量少/价值高的销售策略，将会怎样？

很多伟大的产品构想最终被证明是不切实际的，超前于其所处的时代，或者是无利可图的。能够对相关概念如何互相联系进行分析，将能帮助你质疑产品构想，从而使其更具可行性，或者让你知道何时放弃它。

如果你想让其他人参与进来，商业模式也是有用的。他们可能是雇员、业务伙伴，或投资者，等等。不管是哪一类人，他们期望知道的是，该项业务是有潜力的，已经过深思熟虑，并且有可能生存下去，甚至可能会相当成功。对于初创企业，你没有可以提及的既往成功记录。虽然推介企业主要靠的是产品本身，当然还有你所投入的激情，但对于计划向你的企业投入时间或可能投入资金的人来说，你能否坚持一个制定周密的商业模式是一个重要的辅助考查因素。

也许在一定程度上，顾客也会考虑是否值得在你的产品上投入时间和资金。他们会问自己一些相关的问题。让我们在因特网产品这个更广泛的领域里，大体看一下其中可能涉及的一些问题。

- **我为什么要浪费时间去试用另一个社交网络？我想等等看，看我所有的朋友是否会先加入其中。** 第一个问题是关于你的“价值主张”（即产品）的，如果你正试图进入一个已有好的或流行的解决方案的市场，它就是一个合理的问题。
- **如果你们破产了，这个能接入因特网的机器兔子会不会成为一个昂贵的镇纸？** 这种事情已经发生在了电子宠物兔Nabaztag身上。作为物联网领域最早的消费类产品之一，这些兔子形状的装置能响应经由因特网收到的刺激信号，并据此喃喃自语和移动耳朵，从而博得主人的开心，直到制造它的法国公司Violet破产为止。机器兔的新主人，Mindscape，考虑到了这个问题，因此开源了Nabaztag（和它的后续产品Karotz）的代码，以确保无论公司出现了什么情况，客户们都能继续使用该产品。这个问题的提出，一定程度上体现了消费者在面对商业风险时的精明。潜在的客户已经目睹了其他公司的破产，因而并不想让因此产生的不便甚至浪费发生在他们身上。
- **你们的在线文档协作看上去很棒，但它值得我把所有的业务都迁移到它上面吗？如果你们停止运作或改变平台，我们可能不得不再次重做所有的工作。** 这样的客户可能会对你的商业模式的细节感兴趣，会据此估算风险，并根据他们能辨识出的风险，判断是否值得投入。这不只是对公司生存能力的担忧：谷歌公司停止运作的可能性不大，然而很多企业还是不愿意依靠Google Drive来编辑文档。部分的原因是，他们不了解该产品在谷歌公司中的战略地位，不能确保该服务不会被中断或削弱；或者，他们预料到了一旦该服务终结免费模式后的成本结构。
- **这项免费服务太棒了，但你们为什么不让我为此付费，这样我就可以获得一致服务，得到支持，并避免看到广告？** 最后，很多客户都意识到了应该有可选的他们更喜欢的付费模式，并且可能更愿意选择付费模式。并非所有的客户都会选择免费模式。当社交化的书签网站delicious.com开始失去很多曾让其大受欢迎的功能时，Maciej Cegłowski创建了提供付费服务的pinboard.in。它的付费模式是，每当有一名新用户完成付费后，就把金额不大的注册费增加一点点，这样做的目的是让用户基数保持足够小，使得他的开发和支持能够跟得上。

Cegłowski曾经说过：“你真的无法知道所注册的那个很酷的项目是在硅谷的一幢摩天大楼里，还是像我一样：一个只穿着内裤的家伙，躲在某个地方，同时开着五个连接到终端服务器的窗口。”

（http://www.economist.com/blogs/babbage/2011/04/price_fame）

但是，合作伙伴、投资者和知情消费者想知道这些。尽管，我们已经看到，一个商业企业和只有一个人的团队一样，都能很容易地放弃一个产品，但商业模式的用途之一，就是可以作为一个有用的工具，用来了解在上述两种情况下，企业保持服务运作的计划。

关于“免费”产品，有这样一种说法：“如果你没有为某样东西付费，你就不是客户，而是被出售的产品。”安德鲁·刘易斯（Andrew Lewis）在2010年时，让人们普遍了解了这一提法

（<http://www.metafilter.com/95152/Userdriven-discontent>），但它是建立在很多已有的关于消费的评论之上的，例如，广告克星媒体基金会（Adbusters）于1999年发布的经典视频“你就是产

品”（http://www.adbusters.org/abtv/product_you.html）。不过，可能有一个和这句口号同样简练的疑问：这是真的吗？社交化创业公司Cute-Fight CEO Derek Powazek，对几个常见的假设提出了质疑：

- 不付费意味着不会抱怨；
- 你要么是产品，要么是客户；
- 让你花了钱的公司会更好地对待你；
- 所有初创企业都应该对用户收费。

——<http://powazek.com/posts/3229>

Powazek认为，实际需要吸取的教训是：

你的商业计划不能再是保密的了。人们都非常精明，非常厌倦被折腾得焦头烂额，对初创企业倒闭时失去他们贡献的内容非常警惕，并且对服务条款的突然更改会非常恼火。如果从一开始就公开你的商业计划，就可以在将来避免大量问题。

——<http://powazek.com/posts/3250>

9.4 常见模式

我们已经介绍了生成和分析商业模式的工具——商业模式画布。纵观历史，我们看到商业模式有很多不同的常见形式。我们最好来先看一下一些物联网公司已在使用（或可能会使用）的模式，并且考虑一下这些模式涉及画布上的一些参数。

9.4.1 制造销售

作为最简单的一类商业模式，“制造销售”对物联网当然是有效的。艾德里安售卖为客户定制的Bubblino，初创企业Good Night Lamp正准备加大晚安灯生产，使其成为现货产品。正如你将在第10章中所看到的那样，在商店（包括实体店或网店）里销售的电子产品可能会受到法律和认证（RoHS、Kitemarks，等等）的约束。这是一个需要考虑的额外因素和成本。很多小规模的项目选择以“套件”的形式销售产品，但后续需要一些组装的工作。因为套件的用户群被认为是专业人士和爱好者们，而非普通大众，管理成本可能相对较小。然而，这样的决策不仅缩小了目标市场，很可能也限制了潜在的收益。

9.4.2 订阅

如果不支持重要的因特网组件，物联网装置就不能获得有用的最新内容，从而不能保持最新状态，这样它就没用了。当然，持续的因特网服务也意味着提供者要付出代价——服务器的开发和维护，主机托管费用，在某些情况下甚至还有网络连接成本。在这种情况下，采用订阅模式可能就比较适当了，它能使你收回这些成本，并且可能通过收取服务费用获取持续的利润。很多产品都可以合法地使用这种方式，但也许对于更复杂的内容驱动服务，这种方式会让人觉得更有说服力。为了吹个泡泡，按月为Bubblino支付费用，这种做法似乎有点夸张。但是，BERG Cloud这个能向其Little Printer打印机提供排版精美的新闻和娱乐内容的云服务，似乎才是采用订阅模式的理想产品。按照现在的情况看，内容消费者们不用为了使用BERG Cloud或任何内容订阅付费。内容的发布者将来可能会为了使用某些高级服务而付费。也许这个例子表明，物联网产品采用付费订阅模式目前还没有市场。这

可能意味着这个市场正有待建立。既然人们乐意为音乐服务、企业的群组软件支付订阅费用（当然还有手机），那么这些领域的物联网产品采用订阅模式，可能更能引起消费者们的兴趣。

所谓的免费增值（**freemium**，由“免费的”和“优质的”构成的混合词）模式一直是一种既能鼓励付费客户，又不疏远免费客户的方式。在这种模式下，产品的某些部分是免费的，这能鼓励用户支付额外的费用以获得额外的功能或移除一定的限制。该模式可以与前面两种模式结合起来使用，比如说可以让用户在购买物理装置后终身免费使用相关因特网服务，但也有额外需要付费的服务。

9.4.3 定制化

我们之前谈到过，作为对批量生产的改进，买车的过程可以依照买家的需求进行微调。由于物联网装置跨越了实物和软件两个领域，所以必定会有很多用户定制方式会导致新商业模式的出现。

对于大批量生产的物品，任何定制必须被严格限定在一个清晰定义的范围内：不同颜色的喷漆选择、诸如轮胎、装饰物和内饰等配件的选择，以及车载计算机控制和显示功能的选择。福特式逻辑决定了所有这些组件必须针对制造过程做优化，并且相互间能很好地配合。

相比之下，软件世界具有极高的可塑性，完全有赖于我们的想法。早期的网站通过使用**<blink>** 标记和gif动画图片，把对HTML这种新媒介探索到了极致。然而时至今日，Facebook、Twitter和Pinterest等公司的主页，只是在严格定义的范围内提供了小幅度的定制能力：（雅致的）配色方案选择与头像图案选择，等等。

很多物联网产品具有一定的可定制性：每个Bubblino都有一个名字（艾德里安给起的），而用户也能对推特上想要监听的短语进行更改。BERG的Little Printer打印机不仅提供了对打印内容的选择，也能在等待新的传送内容时，对打印显示的笑脸图案进行选择。（当然，当前只有有限数量的选项可用。）尽管该打印机的输出和为它提供内容的软件一样，十分灵活，但BERG对选项做了限制，以免破坏产品的美感。

新的制造技术，例如激光切割和3D打印，都为物理装置的定制提供了更大的可能性。**MakieLab** (<http://makie.me>) 支持在线设计洋娃娃并将其制作出来。由于是按照你所提供的规格进行制作，它们成为独一无二、完全属于你自己的洋娃娃，这是批量生产所无法具备的。尽管很多组件实际上是从一个有限的列表中选取的（假发、上衣、围巾），但还有一些选项可以用滑动条来修改，例如，眉毛的模样可以选择“不友善”或“凶恶”。有人可能会说，这实际上和我们前面提到的比较表面化的定制是一样的，只是有更多的可调整变量。然而，在某种程度上，这些众多选项导致组合激增，使得创建你的**Makie**洋娃娃的过程看起来像是真正的个性化定制。**Makie**洋娃娃的制作过程组合使用了3D打印、自动化控制和手工组装。虽然**Makie**洋娃娃的当前版本还不是一个物联网装置，但它已经实现了真实世界和虚拟世界的沟通，使在线的虚拟形象成为了真实的存在。在未来，这样的洋娃娃将肯定能跨界成为物联网装置。于此同时，这些有趣的案例还能帮助你了解自己在开发物联网产品时可能用到的新技术。

9.4.4 成为一种关键资源

不是每一个物联网企业都会面向大众市场销售产品。一些企业会向其他公司出售组件或专业技能，即提供组件制造或咨询服务。实际上，从事这类业务时，你把自己定位为他人商业模式中的一个“关键资源”或一个“合作伙伴”。这些商业模式非常有效。诸如**Adafruit**的和**Oomlout**之类的小公司面向爱好者和创客销售电子元器件。制造商为装置（如物联网装置）的生产商生产印制电路板（PCB）和其他客户定制的电子器件。在咨询方面，无论是只提供技能并收取费用，还是为涉足物联网的公司提供战略规划方面的愿景和专业知识，都是可以做的。

这些相当清晰的供应商/咨询关系说明企业和消费者一样需要解决问题。在本书中，我们主要在介绍消费类产品，因为它们最能契合我们“技术要为人类服务”和“物品的地球村”这种观念。然而，企业客户市场可能同样重要。“我们看不到有更多的工程师和企业家对企业感兴趣，这让人震惊。”红杉资本的**Jim Goetz**评论道。他看到有很多面向消费者的初创公司到他这里寻求投资，但面向企业的初创公司明显更容易获得成功。（www.businessinsider.com/sequoiacapital-jim-goetz-on-enterprise-startups-2012-9）

环境数据咨询公司amee (www.amee.com) 不仅为消费者，也为企业和政府机构提供方案，通过获取有关碳排放量的实际数据（不只包括它们直接使用的能量，也包括它们在处置废弃物时使用的能量），使其逐步改善对环境的影响。虽然在很多方面amee是一家软件类的因特网公司，其业务是整理有关各种活动或产品的碳成本数据，但由于他们对环境数据十分感兴趣，所以很自然地跟物联网领域联系了起来。该公司为工厂中生产咖啡豆的过程建立了模型，使得传感器可以实时监测研磨和包装每一包咖啡豆的碳成本数据，并在最终包装好的产品上打上这个数据，以帮助消费者在购买时做出明智的决定。

9.4.5 提供基础设施：传感器网络

传感器数据是物联网中的一个很有吸引力的话题。虽然有官方的数据源，并且这些数据都经过非常精确的校准，而且生成这些数据的成本也不菲，但想获得这些数据可能并不容易。而且毫无疑问的是，政府机构或公司只会把他们那庞大但同时又非常有限的资源用在某些他们感兴趣的地方，从而更加限制了这些数据的范围和数量。传感器爱好者们所提供的第三方数据“长尾”，对官方数据不仅是一种补充，有时甚至能超越官方数据，成为主要的数据来源。我们需要一个能聚合这些数据的平台。Xively (<https://xively.com>)，之前称为Pachube和Cosm) 就是努力承担这一角色的公司之一。它们允许任何客户实时上传传感器数据，例如，福岛核灾难之后日本的辐射水平。并且，它能对来自于很多提供者的数据进行映射、绘制图表和比较。提交这类数据的很多日本工程师和爱好者们使用的都是自制的经过校准的盖革计数器。虽然任何一台装置都有可能存在精度问题，但一旦把所有的数据汇聚起来，真实的模式就会呈现出来，而对于一个官方机构而言，这一点是很难做到的。

Xively自创立伊始，其目的就是为开源数据提供一个免费的公共基础设施。同时，Xively也提供增强的商业产品。它们具备增强的能力、隐私选项和正式的服务水平协议（SLA）。虽然在写作本书时，Xively提供的物联网“中间件”可能还很难盈利，但在不断增长的物联网市场中，能成为基础设施的一个主要参与者，企业会具有巨大的潜在优势。2011年，Xively被作价1500万美元出售给了新主人LogMeIn。后者为这个初创企业每季度提供约100万美元的运营费用，并且正在与

ARM等合作伙伴就合作事项进行谈判

(http://readwrite.com/2011/07/20/pachube_acquired)。

空气质量蛋是一个由Xively赞助，经由Kickstarter募集资金的项目

(<http://airqualityegg.wikispaces.com/AirQualityEgg>)，是一个借助标准化产品创建相同类型的传感器来共享数据的项目。虽然空气质量蛋是一个开源项目，除项目发起人之外，有一个社区负责它的开发，但它也为类似的项目指出一条道路，即可以把社交意图和商业模式结合起来。澳大利亚技术专家Andrew Fisher提出，此类项目要获得成功，必须要：

- 获取信任；
- 分散部署到各个地方；
- 高度可见；
- 完全开放；
- 可升级。

——<http://ajfisher.me/2011/12/20/towards-a-sensor-commons>

一款商业化产品没有理由不去满足所有这些需求，虽然我们肯定会看到一些企业纯粹出于赚钱的目的而接受和延伸这些良好的愿望。传感器数据是信息，它可以被免费共享，或者可以只是被用来出售。很多能源供应商都推出了“智能电表”，从而有可能实现更高的效率以及更便宜的账单，但也因此聚集了大量的信息（特别是在交叉检视不同的数据源时）。我们将在第11章介绍传感器数据引起的伦理问题。对于这个商业模式，你需要考虑这样收集数据的合法性（在现在和可以预见的将来），以及这样做是否符合公司的价值观。

9.4.6 获取提成

对于传感器网络这个例子，如果所收集的数据价值超过了传感器装置本身的成本，你还可以免费提供这些实物形式的产品。实际上，能源公司经常免费提供它们的智能表。你还可以在装置中链接广告以降低

售价。虽然这种做法是有争议的，但很多美国消费者在购买Kindle电子书阅读器时，选择了支持广告的版本以节省初始支出。

前面说过，即便不向物联网装置的最终用户收取费用，还是有很多办法从某处获利（广告收入、来自于其他公司或国家机构的数据服务费用，以及为所产生的数据带宽而获得的佣金）。在物联网这个新兴领域中，到底有哪些能“被出售的产品”仍然是一个有待探索的领域。也许将来的Bubblino版本也可以偶尔被广告商的推广推文触发。也许你的因特网冰箱在你往里面放东西时会发出啧啧声，并为你的下一次采购推荐其他的（促销）选择。

9.5 为物联网初创企业筹资

将来的成本和收益对于一个精心策划的商业模式是非常重要的。与此同时重要的是，将来很可能会存在一段只有支出没有收入的时间。如何获得初始的资金支持是一个关键的问题，值得我们去了解解决这一问题的若干种方式。

如果你有足够的个人资金，能让你把全部时间都投入到你新创办的物联网初创企业，而无需承担额外的工作，你当然可以自己为自己的企业提供资金支持。除了会有砸钱到一个没有真正成功机会的个人项目的风险（本章的目的就是要避免这一点），这将是一种非常幸运的情况。如果你还有多余的钱支付材料和人工费用，那就更幸运了。

如果你不是蝙蝠侠布鲁斯·韦恩，而是和我们大多数人一样，不要担心，你还是有办法启动项目的。如果企业初始阶段不需要巨额资金的投入，你的时间将是主要限制因素。如果你不能够把新项目作为全职工作来做，也许你可以抽出周末的一天或下班后的几个晚上做这件事。你或许可以做出安排，用部分工作时间做这件事，甚至是额外挤出的一个下午或一天可能也足以让项目继续下去。很多人试图把初创项目和咨询业务结合起来，打算获取短期但又利润丰厚的订单，以此来支持下一阶段紧张忙乱的创业工作。保罗·格雷厄姆（Paul Graham）¹ 建议对这种做法持谨慎态度，因为咨询容易赚钱，企业可能会过度依赖它，从而让初创企业的一个主要的源动力——对失败的恐惧消失。在一个措辞尖刻的脚注中，他阐述了这一主题：

¹ 硅谷创业教父，畅销书《黑客与画家》作者。——编者注

开展咨询业务是产品制造公司走向消亡的途径。IBM是最著名的例子。以咨询公司的形式开始，就像是从坟墓出发，然后试图向上走出一条路，通往活人的世界一样。

——保罗·格雷厄姆（www.paulgraham.com/startupfunding.html）

确保你不需要在初创企业上花费巨资是一个关键。在创业初期，你可能不需要办公室，也不需要昂贵的艾龙（Aeron）座椅。你可以在厨房

的餐桌上、在咖啡厅里，或者在共用工作空间中工作。

我们在有关原型制作的章节中讨论的所有内容，就是为了得到一个最小可行产品（**Minimum Viable Product**），来向人们展示，开始吸引公众的注意。使用可以在云端部署应用的诸如**Heroku**等便宜的托管账户或服务、**Arduino Ethernet**板、一些基本的电子元器件、一些硬纸板和一把刀，你就基本上可以完成这项任务了。等到获得投资后，你就能在真正需要时对原型中的任何一部分增加投入。

9.5.1 业余爱好项目和开源

如果项目也是你的爱好，你可能不会有额外的支出，因为反正是在业余时间进行的活动。但是，如果你阅读本章的意图是想搞明白如何把产品转化为一个成功的企业或社区，那么和纯粹的业余爱好所需的时间及节奏相比，你可能就不会那么轻松了。

要想让项目能较快发展，可以用开源的形式来发布项目所有的详细信息，并围绕这个项目培育一个社区。这可能是件苦差事，但在吸引好的协作者以及在与他们维持关系的过程中，你所表露出的天赋，取得的经验，乃至遇到的运势，都会让人受益匪浅。但是，将项目开源之后，你就不能再把它转为闭源了。是的，你也许可以为该项目创建分支版本，并继续秘密地在这个分支上开展工作。但如果你的协作者们对现有项目有足够的热情，它就可以继续发展。实际上，你的想法、代码和原理图都可以被别人用在他们自己的商业产品中。为此，慎重考虑使用哪种许可方式很关键：较严格的许可，如**GPL**，要求那些建立在你工作之上的项目也要以相同的许可条款来共享它们的源码。因此，使用**GPL**可以有助于将从事商业开发的用户群体锁定为那些愿意与你共享成果的用户。此外，当考虑开源时，别忘了，作为项目的发起人和拥有者，在围绕该项目成立一个公司时，你处于最有利的地位，并且更有可能利用自己与社区的关系获取利益，这主要表现在：

- 很多人都在帮你做测试，报告问题，解决问题，以及构建新功能；
- 很多热情的用户为产品提供了真实的使用案例和意见，这比任何讨论小组都有效；

- 社区的友善，以及由此所形成的丰富的个人建议与强大的社交媒体营销网络。

运作一个开源项目要做很多工作。因为容易失去项目控制权，所以这种做法可能并不适合每个人，但它的确是一个值得考虑的选择。

9.5.2 风险投资

当然，从外部投资者处获取项目资金本身需要做一些工作，且会引起一些风险。申请资助的过程需要花费时间。尽管其中大部分时间是在研究制定商业模式，因而可以看作是正当合理的投入，但这些事和你实际想做的产品本身相关的工作没有直接的关系。初创企业通常会集中进行几轮资金募集活动。每一轮中，它们通常为了商业计划中规划的一个阶段，非常努力地让募集的资金达到一个目标金额。

在任何一轮正规融资到来之前，首先是一轮来自于亲友、朋友和傻瓜的（FFF轮，**the friends, family, and fools round**）投资。这个阶段也许是这样的：你拿出了毕生的积蓄，并且基于你的信誉，说服了姑妈、最好的朋友或一个本地的小企业投入其余的资金。虽然这种融资方式可能会对你的人际关系造成一定的影响，但这一轮融资可能是最容易把握的。

常见的下一个步骤是天使轮。所谓的天使通常是指个人投资者。他们大多本身是企业家，并且愿意在更为正式的投资（如马上就要介绍的风险投资人）涉足前，对一些处于早期阶段的初创企业进行投资。他们愿意这么做的原因可能是：这些天使投资人的技术和商业背景可能和你的产品相关；或者，作为个人投资者，他们只是有更多的机会凭自己的直觉判断你的价值。天使投资人的出资额度一般在几万或几十万英镑这个范围内。对于早期的初创企业，这已经算是可观的数额了。此外，天使投资人的个人兴趣，以及他们为你的公司带来的经验，意味着他们的建议、人脉和其他帮助很可能会和他们所提供的资金一样有用。尽管在如此早的阶段，在这些公司尚未能够证明盈利能力之前就进行投资需要承担很多风险，但天使投资人往往会对多家公司进行投资以分散风险。他们通常会要求获得公司的股份，即一定比例的公司价值。如果将来你做得很好，这些股份就能用来回报他们的投资。这些天使投资人也可能会要求成为公司董事会中的一员。这样

做既是为了监管他们的投资，也是出于想帮助这些公司成功的意愿和兴趣。

在美国，寻找天使投资人的一个好地方是AngelList (<https://angel.co>)，它是一个让投资者能与初创企业接触，继而满足长尾需求的信息汇总网站。

风险投资（VC）轮也是类似的情况，但投资方不同于那些需要你努力讨好的个人天使投资人，而是一个拥有大量资金的更大团体，它的唯一目标是发现并投资具有显著盈利前景的新公司。如果天使投资人已经向你投资过，它们可能会对你感兴趣。如果其他的风险投资公司已经在考虑向你投资，它们肯定会对你感兴趣。它们肯定会要求获得股份，可能会要求较高的持股比例，并且会要求成为董事会中的一员。入职董事会的目的还是一样的：既是为了帮助你的管理团队在尚未顾及的地方弥补空缺，也是为了对你和它们的资金进行监管。通常情况下，风险投资公司的出资数额较大，一般从50万英镑起步。

如果你的想法不错，并且团队的某位成员有时间并具备一定能力，你是有机会让VC之间相互竞争从而获得最佳交易的。下面这本书对此有详细描述：*Venture Deals: Be Smarter Than Your Lawyer and Venture Capitalist*。

还有另一个相关方案（特别是在早期阶段），那就是加速器（*accelerator*）。它可能是由风险投资公司运营的。在这种情况下，授予你公司的资金，有部分或全部都是以实物形式支付的，可能会包括免费的办公空间、咨询服务，以及在那些投资人相信会让你成功的领域内给予的特定训练和指导。加速器可能只在设定的时间点接纳申请人（像是一轮一轮地进入），或者可以在任何时候接纳好的初创企业（假定有足够的容量）。与其他充满灵感的新公司被安置在同一孵化器中，可能会有很大的好处，并且所获得的训练和人脉可能很有价值。

当前，专注于物联网领域，或者兴趣领域涵盖物联网的加速器包括：

- HAXLR8R (<http://haxlr8r.com/program/hack-what>) ；
- PCH加速器 (www.pchintl.com/accelerator/accelerator.aspx) ；

- 柏林硬件加速器 (<http://www.berlinhardwareaccelerator.com/>) ;
- Bolt (<http://www.bolt.io>) ;
- Lemnos实验室 (<http://lemnoslabs.com>) 。

在写作本书时，美国本土的加速器正在逐步增加。这个概念已经传播到英国、欧洲大陆和其他地区。每个加速器方案各不相同。一些方案在办公场所、启导和相邻的初创企业等方面的品质可能更为优秀。认真审视每种方案绝对是有好处的，通过获取相关加速器的最新推荐资料，可以确保它们是否适用于你。

Y Combinator (<http://ycombinator.com>) 也是一个加速器。它的创始人保罗·格雷厄姆欢迎硬件初创企业的入驻，并且这样谈及“硬件复兴”的观点。

这股趋势的背后有许多力量在推动。硬件产品在众筹网站上表现出色。平板电脑的普及，使得制造由其控制，甚至使其包含在内的新硬件成为可能。电机也有所改进。现在可以理所当然地拥有各种类型的无线连接。做出硬件产品越来越容易了。Arduino、3D打印、激光切割机以及比以往更易获得的数控铣床，都让硬件的原型制作变得更容易。随着越来越多的客户选择网购，零售商也不再是一个太大的阻碍。

——<http://paulgraham.com/hw.html>

尽管这些投资可能听起来像是“免费资金”，但正如前文所述，获得投资是有一定条件的：投资人会占有股份，并且经由董事会实施一些控制措施。显然要做出一些妥协：你不能用这些资金在其他方面发展你的企业。即使失去一些对公司的控制权会让你痛心，但在有价值的公司中拥有较少比例的股份，要比在毫无价值的公司中拥有较多比例的股份划算多了。

我们已经介绍了一些筹资方案。此外，你需要清楚，接受风险投资的同时，你也对它将来的退出做出了承诺。退出策略是指“风险投资人或企业主打算抽回他（她）所做的投资时所采用的方法” (www.investopedia.com/terms/e/exitstrategy.asp) 。因为投资人会

索取回报，所以长期目标不能仅仅是让公司成功，也要把偿还投资考虑在内。通常情况下，只有两种退出方式可选。

- **公司被大公司收购了：** 在这种情况下，收购方买下了投资者的全部股权。也就是说，收购方根据他们对公司价值的估值和投资者的持股比例，给予投资者相应的回报。公司的创始成员通常会转移到收购方，因为他们是公司的主要资源之一。现在他们已经失去了对公司的控制，为了让他们保持继续为新公司工作的动力，支付给他们的款项通常会有一部分是以股份的形式。这些股份要等一定时间之后才能行权（例如，服务一年之后才允许兑现）。
- **公司通过IPO（首次公开募股）成为上市公司：** 这涉及在股票市场上发行和出售新股。虽然这个选择会“稀释”已发行的股票的价值，但上市之后，现有股票的持有人能够在市场上出售股份，从而收回投资，或者如果他们相信这些股票会增值，也可以保留这些股份。虽然IPO也是一种能够进一步为公司筹措资金的好办法，但在进入股市后，公司就有了新的承诺：季报、新股东的投票权，以及当市场周期性地重新评估你的股票价值时，公布公司运营状况的义务。

显然，这两种常见的退出方式并不适合其他可供选择的商业模式，如合作社，或者像英国的约翰·刘易斯百货商店这样的资本主义员工所有制企业。这些商业模式可能与当代贸易的标准流程大相径庭，只有当众人的经济利益达成一致，这些理想化的替代模式才能存在下去。

9.5.3 政府投资

各国政府通常都想促进本国的产业和技术发展，他们可能会提供资金，以帮助实现特定的目标。试图对全世界的各种资助计划进行介绍本身就是一个项目，因此，我们这里只是基于英国当前的状况，做一些最一般的评述。

虽然政府可以建立它们自己的风险投资基金，或者以各种方式与现有基金进行合作（它们也的确这么做了），但通常它们会以不同的方式管理大部分的资金。原因之一是，它们也想资助现有的公司从事新的研究和创新，而这些公司可能不能接受出让股权的想法。

提供的资金仍然带有“附加条件”，但它们很可能采用不同的处理方式。

- **成果：**可交付成果是一个度量指标，提供资金的机构可用它来判断你是否在做该机构想要资助你做的事情。这个度量指标可能只是一个测试，看你是不是把资金管理得很好，或者可能和该机构本身想推进的目标有关。你可能会被要求定期写报告，或者通过进度表上的某些已定义的里程碑。如果你得到的资金是分阶段提供的，后期的资金支付可能是以成功交付前期成果为前提条件的。你需要非常清楚所做工作，以及这些任务有多么繁重。如果你将要在次要的活动上花费大量时间，即便是一大笔资金也是没有价值的。你可能会被要求提供配套资金。也即是说，如果给你1万英镑资金，你自己必须也募集1万英镑。某些英国机构，如技术战略委员会（TSB），目前就是以这种方式运作的。理解具体的配套方式非常重要：

例如，你可能需要用你的配套资金先行支付，然后再把拖欠的与花费的金额相同的资金拿回。如果提供资金的机构不定期地支付这些资金，例如，按季支付，你可能会在关键时刻无钱可用。因此，你需要了解这个过程，因为可能需要额外筹资和进行现金流管理，或者，从提供资金的机构申请资金时可能需要耗费资源。

（提供配套资金的要求对于大公司非常合理。如果政府提供配套资金，它们就愿意投入一些它们的现金储备做研究。这种方式不太适用于预算紧张的初创企业）

- **开支限制：**一些资助可能会要求在特定项目上给予一定比例的资金支出。例如，商业咨询或Web开发，并且可能还是与基金协调人的公司或联营公司开展这些业务。这一要求当然有可能是极具价值的，但在操作层面看似有些误导作用；倒不如明确澄清这是一种实物形式的资助，而不是现金资助。如果你在运营一家资金紧张的初创企业，你可能能以低得多的成本获得这些服务。你还必须考虑利用这些（可能不需要的）服务所耗费的时间和管理开销。请确保你能了解伴随头款而来的出资方的期望。

在已经介绍过了风险投资以“轮”为单位的投资方式后，你可能会对政府投资在整个投融资体系中的位置感兴趣。投资可能会涵盖可行性调

研，研究开发，以及可能存在的旨在帮助你把原型转化为产品的“生产轮”。从创客或黑客的角度看，“可行性”和“研究”似乎是多余的。这些都是理所当然的事。往往在下一步，即把原型引入生产过程时，你可能需要一些帮助。然而，政府会试图出钱资助那些它们在政策层面感兴趣的成果，并倾向于为研究拨款，而不是为了帮助产品进入市场。这么做也非常合理。毕竟，在产品得到验证后，企业应该有能力自筹资金或获取风险投资。

企业获得多个来源的投资是件非常正常的事。例如，如果已经成功获取政府创新资助的企业，在以后接洽风险投资机构以获取更多资金时，将处于非常有利的地位。

9.5.4 众筹

我们已经把长尾作为一种商业模式做了介绍。众筹可以被看作是投资项目的长尾。让很多人为一个项目出资不完全是一个新现象。1238年，Walter Gervaise通过与好友以及其他富有的市民接洽，获得了公众的捐款，并建造了埃克斯河上的第一座石桥

（www.exeter.gov.uk/index.aspx?articleid=2879）。在此之前，奥古斯都·恺撒在公元前27年之后，为了给他的医生安东尼·穆萨制作雕像，发起一次公开募捐（www.gutenberg.org/files/21325/21325-h/21325-h.htm）。几千年来，很多民间和宗教的古迹和建筑，其建造资金至少有一部分是由公众提供的。然而，这些项目大多数是由一些有影响力的人物或机构发起，并引起公众关注的。借助因特网长尾带来的效率，人们在做各种事情时都可以寻求资助，例如，以限量版的形式出版他们最新的漫画书，发行包含史密斯乐团每一首歌的声乐版翻唱唱片

（<http://thesmithsproject.blogspot.co.uk>），或者是生产激动人心的新物联网产品。

截至2013年，众筹的两个主要的选择是Kickstarter

（www.kickstarter.com）和Indiegogo（www.indiegogo.com）。从历史上看，只有那些位于美国的项目能使用Kickstarter募集资金，而Indiegogo则把自己定位为“全世界的投资平台”。现在来自于英国的项目也能使用Kickstarter了。如果你位于一个没有被Kickstarter覆盖的国家，你可以考虑Indiegogo。另一种办法是在美国或英国找一位核心协作者，但这样做当然会带来更多沟通和组织方面的开销。

目前来看，Kickstarter比Indiegogo的吸引力要稍微大一点。有更多的人听说过Kickstarter，这使得人们为你提供资金的可能性更大。Indiegogo对各种类型的项目都是开放的，包括社区和慈善项目，而Kickstarter只针对那些最终能形成产品的创新性项目，艺术类或技术类项目都可以。鉴于Kickstarter设置了更为严格的限制，对于在Kickstarter上创建项目需要一个申请过程也就不足为奇了。不是所有的项目都能获得批准。Indiegogo扮演了一个明显弱化了了的看门人角色，允许你马上开始宣传你的项目，无需审核流程。

虽然我们已经看到，依照政府基金打算支持的成果进行申请，可能会受到青睐，但风险投资者可能希望了解到你的商业模式是正确合理的，团队是足以胜任的，但使一个众筹项目成功的要素可能更难确定。一个好的概念仍然重要。吸引人的文字、华丽的视频和好的设计能在你的项目和竞争项目之间形成差异。即便是一些筹资成功的项目也可能会失败，因此，资格老的和聪明的众筹投资者们更有可能投资于商业模式受到一定关注的项目，或有成功完成项目记录的团队所负责的项目。

总之，为你提供资金的一方是真实的人，有着任何一群真实的人都会有的各种关注点和弱点。这种与一个庞大而多元化的群体的互动，是这种投资方式有趣之处的一个关键部分：这远远不只是钱的问题。我们已经看到，长尾模式是怎样让有着各种小众兴趣的消费者们找到对满足这些小众需求感兴趣的生产品的。甚至在你为产品投入时间和资金之前，就能通过众筹应用这个模式了。假设各种聚合网站

（Kickstarter和Indiegogo等）能很好地做好它们的工作，它们能让你的物联网产品被相当数量的潜在客户知晓。如果产品没有受到关注，可能按目前指明和宣传的内容看，该产品不会成功。如果项目传播迅速，引起了广泛关注（偶尔会发生这种情况），并且募集到的资金远超目标金额，你就知道你手上有了一个潜在的成功项目。正如从风险投资者处获得资金的同时，会得到同样有价值的指导和人脉机会一样，众筹在募集到资金的同时，也会得到同样有价值的市场研究和病毒式营销的机会。

9.6 精益创业

我们已经介绍了以低预算运营一个初创企业的优点。做到这点所需的思想方法包括只在真正需要时花费时间和资金——保持饥饿和**精益**

(lean) 状态。由硅谷企业家埃里克·莱斯 (Eric Ries) 首创的“精益创业” (lean startup) 的概念正是来源于这一想法。上一节中的项目众筹选项呈现出了这条路线上的一个更吸引人的阶段：仅当项目存在一个显而易见的小众市场时，才会运营这个项目。

很多精益理论的支持者们建议为项目设置一个着陆页，并在该页面中设置一个简单的表单，用来登记兴趣点。这是一件能快速完成的简单事情，尤其是在有众多初创企业 (unbounce.com 和 landerapp.com 等) 在做这件事的情况下。这些简单的页面允许你申请很多项目，并只聚焦于那些有最多反馈的项目上。不过，如果你已经做了一些原型制作方面的工作，并且对某个单一的想法感觉良好，那么，让事情更进一步，在众筹网站上创建项目，可能更为适当。这么做要比创建一个简单的表单有更多的工作要做，但你能从中学到的东西也多得多。

在许多方面，这种“懒惰”——现在只做最少的事，困难的工作推迟到以后再做——也是我们把原型从最终产品中分离出来的原因。分别有专门的时间用来销售你的产品、确保想法可行和制造可销售的产品。如果你正在考虑“精益化”，你应该把这个观念应用到各个阶段。例如，在生产和销售的最初阶段，你应该努力实现一个“最小化可行产品”。它仍然是一个可销售的产品，而不是一个原型，但由于把所有多余的功能都去掉了，所以感觉像是产品最终版本的原型。因为这个最小化可行产品能用来出售，所有最初的努力都是为了制作出这个产品。在此之后，如果你有时间和资金用来对产品、服务和包装等做一些额外的增强和改进，这将其增加更多的价值。但是，为不完整的原型添加这些增强功能是无法形成一个能有效运作的商业模式的。

精益的精髓是能够迭代，只执行那些现阶段必需的任务，让事情进展下去，而不要为了使一切完美而提前投入时间。你的商业模式只是一个假说，不是一成不变的。这一事实能鼓励你对商业模式进行微调，以此对在真实世界中、对你的产品迭代的过程中得到的反馈进行响

应。此类微调被称为转型（Pivot），通常通过改变商业模式的某一部分实现的（回想一下商业模式画布中的某一个矩形框）。

- **放大转型**：只聚焦于先前价值主张的一部分，并把它转化为完整的最小化可行产品。
- **客户细分市场转型**：意识到实际购买产品的人不是最初的目标人群。虽然你可以继续制造完全相同的产品，但你已经把它销售给了错误的人群。
- **技术转型**：要完成的目标和以前相同，但修改实现的细节。虽然你在从工程角度确定最佳产品的制造方式时，几乎肯定要在技术方面对原型做出很多修改，但这种转型将是一个商业决策，为的是改善制造成本、速度或质量。

在莱斯的书中，对不同类型的转型，以及如何最好地应用它们，有更为详细的介绍。

9.7 小结

商业模式是关于如何依据商业利润或其他一些成功标准把一个项目运营好，以及如何为一个特定的用户群开发出解决问题的产品的假说。纵观历史，人们一直在创造商业运营的新方式，而新技术是最有可能带来全新商业模式的因素。作为一种技术模式的转变，物联网将促成完全意想不到的新商业模式的诞生。面对这一确定无疑的趋势，对自己的商业模式进行分析、讨论和迭代正变得越来越重要。但它毕竟只是一个假说，因而在面对已经存在的状况时可以做出改变，而这正是“精益”方法的重要要素之一。

在这个快速发展、互相竞争，对商业和技术都越来越通晓的因特网世界中，能够向潜在的投资者、合作伙伴和客户展示你的商业模式至关重要。商业模式画布是一个有用的共享类别和术语的集合，能为你与这些群体之间的沟通和讨论提供方便。这一点非常重要，如果你正打算把项目拓展为一个产品，把你的设想变为现实要依靠他们的参与。从朋友和家人，到天使投资人、政府基金、风险投资机构和加速器，我们对各种投资方式做了特别介绍。

当然，创造一个产品，需要的不仅仅是资金。为了更广阔的市场，你需要实现从项目原型到产品制造过程的迁移。在下一章中，我们将对在这一过程中涉及的各种挑战进行介绍。

第 10 章 生产制造阶段

你已经建立了一个初始的原型，并把它展示给了几位朋友（或因特网上的其他人），获得了广泛的好评。他们问的最多的一个问题是“我能得到一个吗？”，你可能开始思索，一个新产品是否要在你手上诞生了。

读过第9章后，你可能对于如何避免破产有了一些认识，甚至可能还多少明白了如何从项目上赚取利润（我们希望如此）。不过，在Kickstarter上发起筹资活动之前，你应该考虑一些其他的事项。

你希望能售出多少个装置？即便你只是打算利用业余时间制作几个，但如果你的想法最终获得了巨大的成功，并且你的订单页面获得了海量的访问量，将会发生什么情况？能发生这种情况是好事，但如果你在计算成本时，因为享受焊接了几块单板的乐趣而认为自己的劳动是免费的，那么，你可能需要做一些修正，以获得更真实的成本。如果不这样做，你很快就会发现，当你为了满足需求，每天从早到晚地焊接这些单板时，你会很难享受到其中的乐趣。

如果不想亲手制作每个装置，你就需要找到能替你做这件事的人。把工作外包出去需要额外的成本，但与此同时，你通常需要大批量地购买元器件，从而降低元器件的价格，因此这两项花费往往能相互抵消。如果制造得足够多，自动化带来的规模效益就会发挥作用，这意味着你可以降低售价或增加利润，或者，在理想情况下，能够兼顾这两个方面。

然后就是一些不太明显的事情。为了不让产品在运输途中损坏，或者，当它们被放置在货架上，并且与很多其他的装置摆放在一起时，为了让它们更有吸引力，你是不是需要为其设计一些包装？认证方面要做些什么工作？如果在美国销售产品，联邦通信委员会（FCC）需要确保产品的安全性，并且不会产生很大的不必要的电磁干扰；而在欧洲销售，则需要在满足类似要求的基础上使用CE标志，并且还需要满足一些额外规定，如危害性物质限制指令（RoHS），该指令禁止销售某些物质（如铅和镉）含量超过一定水平的产品。

在本章中，我们将深入探讨原型转变为成品的整体过程，而且将兼顾明显与不太明显的步骤。

10.1 你要生产什么

在我们开始详细讨论怎样扩大产品的生产规模前，先考虑一下你到底要生产什么产品。

根据你的动机和对产品的期望，暂不考虑你能投入多少时间和资金，你将面临各种可能的选择。

如果你只是想让其他人也能够照你的制作步骤原封不动地把装置做出来，那么只需要记录该装置的构建过程，并把它连同你使用到的任何源代码和设计文件分享到因特网就可以了。你既可以将其发布到你自己的博客或个人网站上，也可以发布到类似Instructables

（www.instructables.com）这样的网站上。这样做了以后，除了可能需要在诸如Make（<http://makezine.com>）或Hack-a-Day

（<http://hackaday.com>）这样的网站上做一些推广，以帮助人们找到这些资源外，你基本上已经实现自己的目标。

大型的创客社区，不管是线上还是线下类型的，都采用这种方式分享他们的项目和经验。加入其中是结识其他创客和回馈社区（如果他们帮助你入了门）的好办法。你可能会发现，你成为了某个项目中不可或缺的人。人们可能会委托你为他们构建相关的项目，或者想聘请你到他们的公司工作。一个实现度较好的项目在哪里都可以作为最好的简历。

当你下定决心，想让你的发明能被更多的人得到（并且希望能在此过程中赚一些钱）时，通常你面临以下三类选择。

- 制作套件：你的客户能自己组装。
- 制作一块装配好的电路板：用户可以在他们自己的项目中，把它用作一个子部件。
- 制作一个功能完备的产品：外壳、说明书，甚至外包装都一应俱全，就等着被摆放到当地百货公司的货架上了。

上述的每种选择都直接建立在前面的工作基础之上。一个完整的装置包含一块组装好的电路板，而制作该电路板则需要一块印制电路板（**PCB**）。该印制电路板又和一个套件中用户需要上面焊接元器件的板子很像。

在本章中，我们将利用这一递进方式，对制造过程的各个不同方面进行介绍。虽然**PCB**相关的小节主要针对套件，但如果你想更进一步地制作，这些内容同样是相关的。对于制作装配好的单板和成品，你都需要了解**PCB**装配相关的内容。

10.2 设计套件

把想法作为一个产品推销的第一阶段就是以套件的形式提供产品。虽然你可能会认为，订购相关的零部件，然后依照原理图把它们摆放好，这个过程是很简单的，但你可能低估了在搞清如何制作套件的过程中，你已经掌握的知识量。的确，有人会跟你一样，最终能构建好某个东西，但更多的人实际上会半途而废，或者因为觉得太难，甚至都没有开始去做。一旦当有人替自己选好并采购了恰当的零部件，很多人就会愿意购买套件，按照步骤指南一步步地将它们组装好。

大多数套件往往只针对特定的应用提供电子元器件和软件，而并不提供任何外壳构件。究其原因，部分是因为，对于作坊式的套件制作产业，采购定制的塑料部件有一定困难。此外，也是因为此类套件的主要市场是创客社区中的其他人，而这些创客们可能更喜欢把套件整合进他们自己的项目中。不过，随着3D打印机和激光切割机的应用越来越普遍，即便是在套件中提供外壳和其他的结构件也正变得非常可行。

套件往往会和现有的微控制器配合使用，通常会采用标准规格的插接板形式，例如Arduino生态系统中的盾板或BeagleBone上的cape板。这样做是有道理的，因为可以减小套件提供商在支持方面的开销。不管用户是否熟悉该平台，他们都可以在其他地方获得很多的帮助内容，从而轻松上手。如此一来，套件的附带文档就能专注于项目构建方面的专业内容。

既然你已经构建出了原型，你就已经完成了创建一个套件所需的大部分工作。你已经搞清楚了需要哪些元器件，去哪里采购它们，怎样把它们连接起来形成一个能工作的电路。你已经编写好了必要的软件，实现了与电子电路单元的接口和对它的控制。你可能还没有完成的部分包括设计PCB、记录构建过程和核算成本。

如果你已经好久没有构建过电子电路，那么，设计PCB时可能会涉及若干新术语和概念。同样，设计完成之后，你可以用几种不同的方式制作PCB。因此，在本章的后续部分，我们将更详细介绍设计和制作PCB。

记录构建过程并不太复杂。当你把一个套件的全部零部件都聚拢到一起后，亲自把它们一步步装配起来。在装配过程中，记得为每一步拍照或拍摄视频，并记下各步骤的次序。**Instructables**网站

（www.instructables.com）上有许多帮助你解决问题的指南文档。另一个好办法是复制其他套件制造商所使用的有用步骤。例如，我们喜欢**Adafruit**为它的每一个产品整理的使用教程。**Adafruit**的**Arduino**电机盾板就具有代表性（<http://learn.adafruit.com/adafruit-motor-shield>）。

制定适当的销售价格则显得更困难一些。这与其说是一门技术，不如说是一种技巧。但有一些经验规则能帮助你。最显而易见的做法是了解你的成本是多少。应该创建一个包含产品所有零部件的表单（至少是一个列表），同时要列出它们的购买成本。你应该列出每一个电子元器件、连接器、线缆、**PCB**、外壳，以及发货时用的包装箱和放置所有零部件所花费的时间。这个列表被称为物料清单（**BOM**, *bill of material*），它是所有成本计算和定价的基础。

物料清单给出了你的产品的边际成本，即在正常的生产过程中，额外再制造一件产品所增加的成本。它不包括建立生产过程或开发软件所产生的任何的固定成本，因为这些成本将被分摊到销售的所有产品中。你仍然需要使定价涵盖固定成本，但明确规定每件产品分摊多少固定成本是比较困难的，因为这将取决于你能卖出多少产品。

最后把拟定价格（你将向消费者收取多少钱）定为**BOM**总成本的4到5倍。这种价格为你提供了盈余，用以包括分摊到每件产品上的固定成本以及一些利润。它也为经销商提供了足够的盈余，用以支付他们的固定成本，并赚取一些利润。这意味着，如果产品最终能够大获成功，可以招募一些中间商来帮助扩大销售，并且不会造成每卖出一个就会赔钱的局面。

假设有多个套件，最重要的可以降低的成本就是**BOM**成本。降低**BOM**成本能给价格的设定带来极大的灵活性，这样就能将利润最大化。

既然已经知道了制作套件的成本以及需在其中加入的盈余，现在就可以考虑售价了。你可以对产品收取多少钱，将取决于人们对它的需求有多少。不过，通过参考类似产品的售价，你能对市场有所了解，并

用这些产品的售价指导你的定价。你不必一定要找到具有相同功用的产品，可以看看那些交付形式类似的产品。例如，如果套件是一块 **Arduino** 盾板，可以把它与其他盾板套件的售价做个比较，从而对人们预期会支付的售价范围有所了解。

因为由客户负责把套件中的零部件装配和焊接在一起，那么通常唯一需要定制的部件就是 **PCB** 裸板了。这往往能把成本保持在较低水平，因为你已经把构建电路所需的工作转移给套件的用户了。对于套件所针对的目标人群，需要自己装配可能会被看成是一项优势，因为这样让套件运转起来能增加成就感和归属感。不过，这也会增加支持方面的开销，因为你将需要通过远程方式帮助客户调试出现的问题。这些问题可能是他们焊接不当造成的，并非套件本身存在缺陷。

所有这些问题都可以通过向消费类产品迈进一步来解决，向客户销售已经完全装配好的、所有元器件都布置到位的 **PCB**。如果销量不会太大，并且设计方案中也没有包括任何特殊的、引脚间距很小或非常复杂的表面贴装元器件，你就可以自己装配它们。否则，你就需要找一家封装厂替你做这件事。此外还应该制定出一个检测成品单板的方法，检查元器件的焊接质量是否良好，最好再检查一下它们是否能正常工作。无论选择上述哪种装配方式，这样做都是有好处的。

显然，从套件到消费类产品的最后一步，是对外壳、联动装置和成品中所需的剩余部件进行制作和装配。因为成品中包含了装配好的 **PCB**，并且增加了更多组件或工艺流程，我们将借助成品的形成过程组织本章后续的内容。

10.3 设计印制电路板

现在你已经做好一个漂亮的原型，或者也有可能它是你做的第5个、第50个或第500个原型，并且你正在筹划下一步做什么。也许最终的原型仍然是插在若干个面包板上的一大堆元器件和连线，或者，你可能更进一步，把它们焊接到了万用板上，正如第5章所介绍的那样。

用焊接的方式固定元器件，可以让原型变得更结实耐用了。如果焊接技术过关，这种连接方式可以使原型具有牢固的电气连接，不会因为偶尔的震动就松脱，从而就实现了一种可以在交付最终用户后仍能继续使用下去的原型，而不会像面包板原型那样，不得不戴着羊羔皮手套小心翼翼地操纵它。因此，这意味着对于每一件正在构建的物品，只要重复这一过程就可以了，对不对？

你可以这么做，但你会很快对自己亲自手工焊接每件物品感到厌倦。接下来可能就需要大量招人，由他们来焊接一个个的零部件了。

于是，制作更为专业的PCB，成为了一个相当自然的进展。

超越万用板这种阶段，转而设计并蚀刻自己定制的PCB，就能实现更多的电路布局方案，并且也能更轻松地焊接元器件，因为板上只有用来插入元器件的孔洞。原本那些不能轻易放置到万用板上网格状孔洞中的元器件，包括一些简单的表面贴装元件，现在也能使用了。

虽然已经前进了一大步，自制的PCB仍然缺少非常专业化的后续处理。这是因为它们没有绿色的阻焊层，也没有丝印层，而PCB制造商会提供这些特性。采用专业化方式制造的PCB能进一步简化装配过程，因为阻焊层会让焊接更加容易，而且更重要的是，丝印层提供了每一个元器件应该放置的位置的轮廓。

如果你在装配PCB，或者将其作为套件中的一部分销售，你几乎只会采用通孔元件，因为对于初学者来说，它们是最容易焊接的。对于偶尔用到的表面贴装元件，你可能可以幸运地完成对它的焊接，但只有当引脚不是太细或太密时才能做到这一点。

其他一些问题会有效地迫使你选择定制**PCB**：如果元器件之间的连接路径特别复杂，只有多层**PCB**能允许连接线之间的交叉；如果任何一些元器件只有表面贴装这一类封装形式可用，定制的**PCB**则允许在不使用额外分接板的情况下放置它们；如果已经在使用现成的微控制器板（如**Arduino**或**BeagleBone**）提供处理器等功能，一块定制的**PCB**让你可以选择把微处理器等远见归并到你的电路板上，从而不再需要单板之间的连接器或者现有单板上用不到的任何元器件，从而节省**PCB**上的空间和零部件费用。

你可以选择多种**PCB**定制方案：可以自己蚀刻（或铣削）电路板，也可以从很多批量制作**PCB**的邮购服务中选择一家，或者让制造商制作好**PCB**并完成元器件的装配。无论选择哪种方案，第一步都是要设计**PCB**。在介绍一些**PCB**设计软件之前，先来看一下**PCB**的构成以及可能会接触到的一些术语。

PCB由若干层的玻璃纤维材料和铜箔构成。这些层互相交替地夹在一起形成了电路板。玻璃纤维不但构成了**PCB**的主要基底，也在不同层的铜箔之间充当绝缘体的角色，而铜箔形成的“导线”把电路中的元器件连接了起来。

考虑到你不会希望同时把所有的元器件都互相连接起来（如果在整个电路板上覆盖一层完整的铜箔，会出现这种情况），铜箔的某些部分可以用蚀刻的方式去除，通常采用化学方法去除，但对于简单的电路板，也可以使用数控铣床。由余下来的铜箔构成的路径被称为**走线**（**track**或**trace**），它们构成了元器件之间所需的连接。

PCB上的走线和元器件的引脚交汇的地方是一些被称为**焊盘**（**pad**）的点。对于表面贴装元件，它们就是电路板正反两面上的一片铜箔覆盖的区域，提供了用来焊接元件的位置。对于通孔连接，焊盘上也会钻一个通孔，用来插入元件的引脚。

单面**PCB**上只有一层铜箔，通常位于电路板的反面。这是因为，单面**PCB**常常用于采用通孔元件的自制电路，而元件会被放置到正面，其引脚穿过通孔后，被从反面焊牢。双面**PCB**有两层铜箔，分别位于正反两面。更复杂的电路，特别是当你开始使用更先进的采用更小封装的处理器时，可能还需要更多的层，以留出空间让所有的走线都能被

布设到需要的地方。3层或5层的PCB并不少见，甚至一些非常复杂的电路都用到了7层的PCB。

当电路板的层数超过2层后，用于放置铜层的电路板的表面就不够用了。额外的层需要更复杂的制造过程，需要采用铜层和玻璃纤维层交替层叠的方式构建，有点像制作三明治的过程。

这意味着中间层被嵌入到了电路板内部，因而就没有了为其设置焊盘的可用区域。对于通孔元件，不难与某一个中间层建立连接，因为引脚穿入的通孔贯穿了每一层。当通孔被镀铜（在孔的内壁覆盖一薄层铜的工艺流程）后，任何在这个位置有镀铜的层都被连接在了一起。

当你需要把两层的走线在某个位置连接起来，而在该位置没有可插入元件引脚的通孔时，你可以使用**过孔**（via）。这是一种与通孔类似，但通常会更小的贯穿电路板的孔，只用来在镀铜后连接不同的铜层。当你不想把每一层都连接起来时，还可以使用**盲孔**（blind via）。盲孔不会贯穿电路板。然而，正因如此，盲孔使得PCB的制作过程更加复杂了。如果不是绝对必要，最好避免使用盲孔。

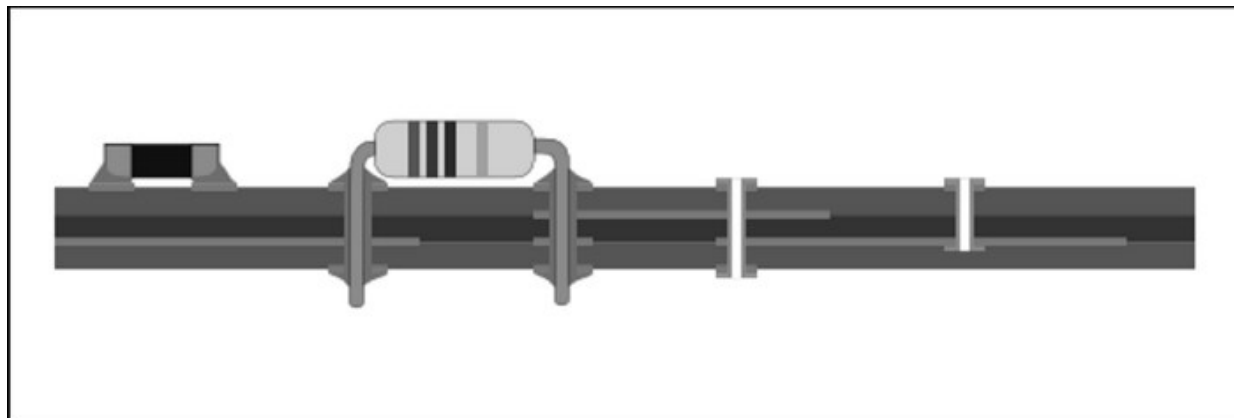


图 10-1 PCB的特性（从左到右）：表面贴装元件、通孔元件、过孔、盲孔

当你需要在一个公共点做很多个连接时，不要在电路板上布置很多走线。可以选择更轻松的方式，把某一层的大部分空间用来做连接，并且只需保留该区域的铜箔即可。这样的区域不叫走线，而是被称为**平面**（plane），常被用来提供接地的路径。这种设计的一个额外的好处是，地平面提供了一个“捕获”某些杂散电磁信号的办法（特别是一些

高频信号线产生的此类信号)。这将减少电路本身产生的电磁干扰,有助于防止与电路的其他部分,或附近的电子装置产生问题。

专业化制造的PCB的表面还要经历额外的工艺流程,要做两件后续处理工作。这使得它们更容易使用。

首先,除了需要焊接元器件引脚的位置,PCB上所有的部分和裸露在外的铜箔都用阻焊油墨覆盖。虽然有其他的颜色可用,但最常见的阻焊油墨是绿色的,而这也是大多数电路板的常规颜色。此类油墨提供了一个阻焊区域,从而让焊剂从这些区域流走,转而附着到那些需要把元器件连接到走线的位置。这样就减小了电路中两个不该有连接的点被焊点意外连接起来的可能性。

其次,在阻焊油墨之上是丝印。顾名思义,丝印是通过丝网印刷的方式涂敷颜料的表面后处理工艺。丝印被用来标示元器件应被放置的位置,而标记位置的目的也是为了便于识别元器件。丝印通常也包含一些细节信息,如设计该电路板的公司或个人的名称,用来描述其用途的名称或标签,以及制造日期或设计的版本号。最后一条信息至关重要;在你排除错误的过程中,很有可能需要对电路设计做几轮迭代。能够从你工作台上的若干PCB中分辨出不同的版本是非常有价值的,或许,更为重要的是,当用户报告了故障产品,能够准确定位所使用的到底是哪一种设计。

10.3.1 软件选择

关于PCB设计软件,我们的选择很多。如果你与某家电子设计公司有合同关系,该公司的工作人员很可能使用的是类似Altium Designer (www.altium.com/en/products/altium-designer) 这样的软件,但你更有可能从下面介绍的低端(也更便宜)的软件中选择其一。Fritzing (<http://fritzing.org>) 是一款特别面向PCB设计初学者的免费开源设计软件包。该软件特意从一个设计画面开始。该画面看上去是一块面包板,允许通过模仿真实的原型制作,绘制自己的电路。然后,该软件可以把这个设计转换为一个电路原理图,允许在PCB视图上布置元器件和调整走线的路径。

对自己的设计满意后,就可以将其导出来制作PCB。Fritzing甚至提供了制作服务,让你的设计更加容易成为现实产品。

还可以把面包板设计视图以图片或PDF文件的形式导出，这令Fritzing更加受到制作项目文档的人的欢迎。即使不打算设计PCB，只想展示面包板上元器件的连线方式以便人们模仿，最简单的一种方法是在Fritzing中重新创建电路，然后把结果保存为图片或PDF文件。

KiCad (www.kicad-pcb.org) 是另一种开源产品，但采用了更为传统的工作流程。它具有更为全面的预定义零件库，可被用来设计具有多达16个铜层的PCB。与之相比，Fritzing只能生成双面PCB。

艾德里安使用KiCad设计了Bubblino的PCB。在第5章的“电路板之旅”补充资料中已经展示了这块PCB。与下面将要提到的两款商业产品相同，KiCad添加的一个最新的功能是查看设计的三维视图。该功能除了让你能看到最终完成的PCB，还可以让你导出3D模型。这样做使你能够在设计外壳时，把PCB的3D模型导入CAD软件，以确保所有组件间有适当的间隙。

在爱好者和半专业化市场中，最流行的PCB设计软件可能是CadSoft公司的EAGLE (www.cadsoftusa.com/eagle-pcb-design-software/)。它之所以能够流行，最有可能要归结于它有一个用于非商业用途的长期免费版本，可以让初学者上手。这导致大量有关EAGLE的操作指南和其他有用的资源被用户社区开发和共享了出来。所以，EAGLE是一个学习PCB设计的不错选择，但除了非商业授权外，免费版也被限制为只能设计二层的PCB，并且PCB的最大尺寸被限制为100毫米×80毫米。

在商业化的产消合一（prosumer）市场，EAGLE的最新竞争对手是DesignSpark PCB (<http://designspark.com/page/designspark-pcb-home-page>)。它是由电子产品分销商RS Components公司提供的，被看作是该公司的DesignSpark社区的一个部分，因此它是免费的。与EAGLE不同的是，DesignSpark PCB没有限制你可以设计的PCB的尺寸（最大达1.2米）和层数（最多支持14层）。不过，它是本节介绍的唯一一个没有Linux或Mac版本的软件。RS Components已经承认兼容性是一个问题，并且的确在努力确保它能在Linux上的Windows兼容层Wine中或在Mac上的CrossOver中运行。然而在实际使用时，这么做是有点笨拙了。

10.3.2 设计过程

不管你决定使用哪一个程序设计PCB，用来创建设计的任务都是被分割到了原理图和PCB这两个主要的视图中。

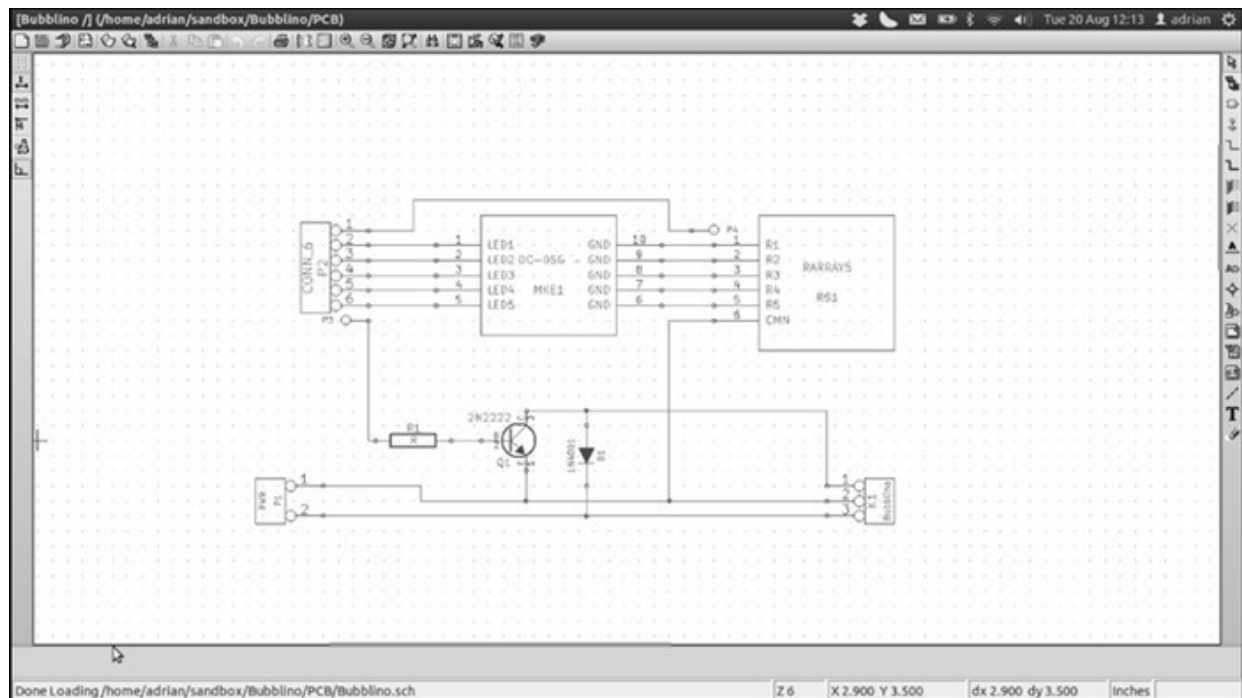


图 10-2 在KiCad中设计Bubblino的PCB时的原理图视图

原理图

你通常是在原理图视图中开始做设计。该视图使你能够按照一定的逻辑布设元器件，建立必要的连接，并且无需操心它们到底应该被放置到物理空间中的什么位置，或者是否存在走线交叉。

伦敦地铁地图没有采用严格按照地理位置布局的方式，而是用易于理解的方式，展示出了站点和线路之间是怎样相互关联的。与此类似，原理图提供了一个概念视图，用来表示电路是怎样布局的。元器件被表示为标准化的符号，并且你经常会按照功能分组，并把它们放入相应的通用功能区。这种分组方式不一定要和它们在物理布局上具有的分组一致。

你所使用的软件包的元器件库中，包含了你需要用到的大多数的常用元器件，如各种固定阻值的电阻、二极管、电容、三极管和集成电路（IC）等。在原理图中添加一个上述的元器件，只需找到相关的元器

件就可以了，有时需要你输入准确的元器件编号。例如，对于一个电阻，你需要标明它的阻值，也有可能需要标明它的公差值。

你还应该确保选对了元器件封装类型。这是元器件的物理形式。很多元器件有着不同的封装类型。选用哪一种要取决于目标应用。制造商的数据手册会列出某个元器件可用的封装类型及其区别。通常会依据焊接方式（通孔或表面贴装）和若干其他的标准来选择封装类型。随着不断实践，你会逐渐熟悉一些常用的封装类型。比如将来不用查询你也会知道，**TO-92**是一个半圆柱体形的有三个引脚的通孔封装类型，通常用于三极管；或者，**DIP-16**是一种16针的通孔IC封装类型，其引脚被分成两列，每列有8个，而**SOIC-16**是一种引脚排列方式类似的表面贴装IC封装类型。

封装类型的选择实际上并不会对原理图造成影响，因为在这个阶段，你只需关注元器件的功能属性，而不是它的物理属性。然而，如果在最初选择元器件时就做出正确的选择，将来当你的设计进行到下一阶段，开始使用**PCB**视图时，就能避免返工。

有时你会发现，你想使用一个元器件，但它并没有包含在你所使用的**PCB**设计软件的元器件库中，这种情况并不少见。如果运气好的话，你可能会发现有人已经以第三方库的形式提供了该元器件。对于**EAGLE PCB**，这种情况十分常见。该软件在开放硬件运动中非常流行，所以人们十分乐意分享他们所用的元器件。否则，你将不得不自己来设计。这一设计过程并不困难，因为你需要的所有信息在数据手册中都有，但要注意细节，因为只要这样才会产生好的结果，使元器件在成品**PCB**上的装配效果令人满意。

当你对单板的原理图设计感到满意后，你可以继续设计物理单板。虽然用这种方式设计**PCB**最合理，但不是一个需要严格执行的过程。你可以在两种设计视图之间来回切换，但在着手设计物理单板之前，尽可能好地完成原理图是有好处的，因为如果设计发生改变，需要重做的工作量将会是最少的。

PCB

关于怎样对**PCB**上的元器件进行布局，并没有硬性的规定。让一组的元器件保持在一起，每组构成一个功能区，如供电功能区，从而形

成一个更合乎逻辑的布局，这是合理的做法。你可能会发现设计中存在一些不能改变的环节，如连接器和排针的位置。连接器都需要沿着一侧布置，为的是在PCB被置入一个外壳后仍能方便使用；布置排针的位置要和Arduino板匹配。除此之外，就是要以最容易走线的方式布置元器件了。首先放置好具有最多连接的元器件，然后围绕它们布置其余的元器件。

当你四处移动元器件时，你可能会注意到，有一些纵横交错的细线把一些元器件连接了起来。这些细线看似和粘连到一块比萨上的奶酪丝一样凌乱。它们把元器件所在位置的各种焊盘相互连接了起来。当你确定元器件的位置时，你应该尽量减小这些细线的缠结程度（通过旋转或四处移动元器件以获取最佳的组合），但不要过于担心这个问题。你只是力求为后续工作（确定适当的连接路径）创造一个好的开端。

这些纤细的点到点的直线连接被称为**预拉线**（air wire）。它们显示了需要实现连接的位置，但如果你不把它们转化为PCB上真实的走线，它们就不会出现在最终的PCB设计中。

每一条预拉线都需要被转化为PCB上的一条走线，并且要在PCB上选择适当的路径，使得这条走线既不会和任何其他的走线交叉，也不会和其他元器件所在的焊盘离得太近。如果两条走线，或一条走线一个焊盘，离得太近，制造过程的缺陷有可能导致在不该有连接的地方有了连接。通过为走线留出足够空间，并不让它们互相交叉，你可以让PCB的不同层发挥作用。把走线设置到PCB的不同层上，意味着当它们之间发生交叉时，并不会产生电气接触。而且，对于更为复杂的设计，你可以在某一层上设置一条走线，把它作为某条路径的一部分，然后经由过孔把走线切换到另一层，从而可以继续实现剩余路径。

你用的PCB软件具有自动布线的功能。你可以让它替你为所有的走线选择路径。然而，此类功能还远不够完善，在实践中，如果不打算完全采用手工布线，你会发现最好的做法是，至少首先采用手工方式布设比较重要的走线。

当所有的走线都布设完成后，设计就基本完成了。应该在丝印层上添加一些标记，用来对不明显的连接器等作出说明，并用来识别PCB本身。此外，版本号也应包括在内，这样你就能区分任何将来的版本。

为了让人们能了解更多，可能还应该添加你的名字或公司名，或者加上网址。

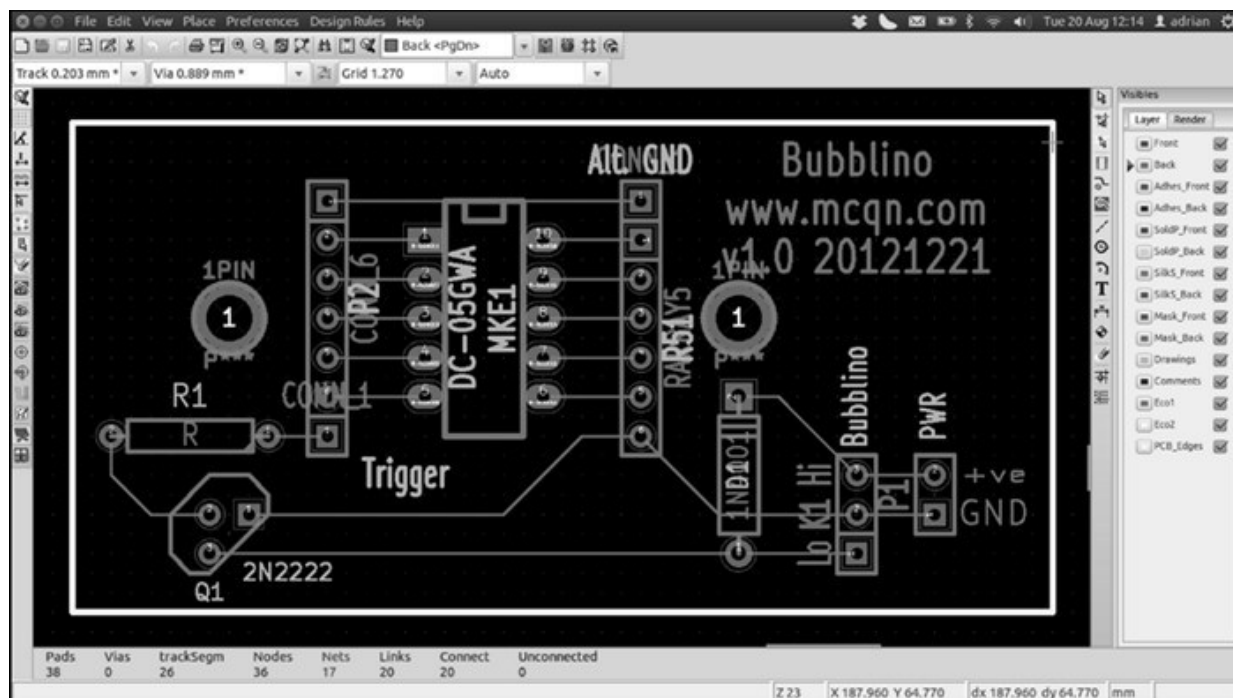


图 10-3 完成后的Bubblino PCB视图，已准备好导出

然后就是对你的PCB做最终的检查，即执行设计规则检查，以发现任何遗漏的连接或与走线相关的问题，如走线互相之间靠得太近，或走线太细，以至于PCB制造商不能可靠地制造它。设计规则实际上包含了PCB制造商的制造公差在内。像最小走线宽度或焊盘之间的最小距离这些参数都属于它们定义的内容。最后，把PCB视图打印到纸上。这样一来，你就能在真实的环境中对元件和它们在PCB上的位置进行比较，发现任何和元件封装有关的错误。

把由这些最终的检查暴露出来的所有问题都修正好后，你就可以制作PCB了。

10.4 制作印制电路板

PCB设计好后，下一步就是制作单块或批量制作PCB。如果只是需要几块PCB，或者想在订购几百或几千块PCB之前先制作几块测试用板（非常明智的做法），你可能会决定自己制作。

10.4.1 蚀刻电路板

最常见的适合家用的PCB制作技术是蚀刻电路板。一些现成的套件提供了所需用到的全部东西。第一步是把PCB的设计转移到将被蚀刻的电路板上。这通常要在PCB设计软件中把设计打印到模板（Stencil）上。如果使用的是光阻板，设计将被打印到一个模板上。在把电路板暴露到紫外光之下时，该模板会把相关的区域遮挡住。如果使用的是碳粉热转印（toner-transfer）法，设计将被激光打印机打印到具有光面的转印纸上。

然后需要把模板上的设计图案转移到电路板上。对于光阻板，你要让它在明亮的灯光下暴露几分钟，而对于碳粉热转印法，则需要使用一个高温熨斗。

在电路板上做好适当准备后，就可以把它浸入蚀刻液了。蚀刻液中的酸性成分会腐蚀掉暴露在外的铜，只留下走线。

腐蚀掉所有不需要的铜箔后，从蚀刻槽中取出电路板，清理残留的蚀刻液。这样电路板就几乎可以用了。

最后一步是在所有安装点或通孔元件所在的位置钻孔。可以人工钻孔，或者如果有数控铣床的话，可以从PCB设计软件包中导出钻孔文件，为铣床提供钻孔位置。

10.4.2 电路板的铣加工

除了使用数控铣床在PCB上钻孔之外，还可以用它把走线四周的铜箔铣掉，从而形成走线。要想实现这一点，需要从PCB软件中以Gerber文件的形式导出铜层。该文件格式最早是由Gerber系统公司定义的，

并由此得名。现在它已经成为了在制造阶段用于描述**PCB**的工业标准格式。

为了把你的**Gerber**文件转换为铣床所需要的**G-code**代码，需要使用另外的软件。（更多有关数控铣床和**G-code**的介绍请参阅第6章。）一些数控铣床已经提供了这个软件，或者也可以使用第三方的程序，如 **Line Grinder**（www.ofitselfso.com/LineGrinder/LineGrinder.php）。

铣削的过程实际上是围绕每条走线的边界铣出一条通道，为的是把这条走线和铜箔的其他部分隔离开来。因此，铣削出来的**PCB**看上去和蚀刻出来的**PCB**有一点差异，因为没有被连接到任何地方的大面积铜箔会先被留在电路板上（为的是节省铣削这块区域的时间）。

10.4.3 第三方制作

如果设计的是超过两层的**PCB**，如果需要一个更专业的成品，或者只是嫌麻烦，不想自己制作**PCB**，有很多公司可以替你制作。

制作**PCB**的价格会依据设计的复杂度和尺寸而有所不同，但不同公司之间也会有很大差别，因此在决定选择哪一家之前，最好先咨询一下，比对各家的报价。

如果急需得到电路板，本地公司往往是最好的选择。通常，这种公司从订货至交货的时间是以天为单位计算的。如果能忍耐更长的时间，则可以委托给位于中国或更远地方的公司去加工。这样做虽然可以大幅度降低成本，但也意味着在收到订货之前，要等待数周时间。

无论你选择的是哪一种公司，你都需要把**Gerber**文件提供给制造商。请确保已经从设计中导出了所有相关的层，包括所用的每一个铜层，外加阻焊层（决定了**PCB**的颜色，阻止任何的焊料附着到不该有焊料的区域）、丝印层（显示标记、版本信息等）和钻孔文件。

10.4.4 装配

PCB制作好后，还需要把元器件焊接到它们上面。

如果是以套件的形式进行销售，客户将会把其焊接起来，所以只需把所有的零部件装入袋子里，让客户做完剩下的事情；否则，你必须负责焊接它们。

对于小批量的情况，可以手工焊接。对于只有通孔的电路板，准备好电烙铁就可以了。表面贴装元件的装配则要更复杂一些，但也是完全可以实现的——只要你使用的此类元件没有采用特别复杂的封装类型。

为了能装配具有表面贴装元件的电路板，你需要**PCB**设计对应的**Gerber**文件集合中的另一个文件：焊膏层文件。你需要用该文件制作一个可以在上面涂敷焊膏的模板。你可以采用激光切割的方式，用一个聚酯塑料薄片制作，或者用薄钢板制作。显然，钢制的模板更经久耐用，并且能够在替换它之前，焊接更多的电路板。

用于焊接表面贴装元件的焊料是膏体状的，采用瓶装或管装的供应形式。你需要借助一把刮刀和一个焊膏模板，把一层焊膏均匀地涂覆到所有元件所在的位置，然后再小心翼翼的把模板从电路板上揭下来。

现在到了棘手的地方。使用镊子把每一个元器件放置到**PCB**上相应的位置。在这一过程中最好能使用目镜或放大镜。焊膏能在一定程度上让元器件固定在正确的位置，但要注意，这时不要敲击电路板，以免一些元器件移位。

当把所有的元器件都放置到电路板上后，需要融化焊料，把元器件固定到位。你可以使用电烙铁做这件事，但如果使用热风拆焊台的话，可能就更容易完成。这种焊台有点像是电烙铁和喷灯的混合体，使用热空气提供所需热量。

如果使用回流焊炉，就可以一次性地焊接所有的元器件。如其名称所揭示的，这个焊炉能对**PCB**和元器件进行均匀加热，直到焊料被融化。专业的回流焊炉可以设置不同的温度曲线，因而对于有更严格要求的元器件，允许你对制造商的数据手册中的规格说明进行匹配。

不过，对于大多数应用，温度曲线的形状不是至关重要的。对于业余或半专业的用途，可以使用烤箱或家用电炉来提供适当的热源，关于这方面的资料有很多，可以轻松找到它们。创客社区的零售商

Sparkfun 有一篇很好的教程，其中介绍了该公司用于表面贴装元件焊接的一些技术（<https://www.sparkfun.com/tutorials/59>）。

当你发展到一定程度，以至于手工装配已经不能满足需要后，你可能会需要用到装配机器人。这种机器人能用微小的真空吸嘴拾取元器件，能够旋转元器件，并将其放置到 PCB 上正确的位置。它们能够以每小时装配数万个元器件的速度重复这一过程。这些机器人被称为**拾取-贴装设备**（pick-and-place assembly machine）。

此类设备的价格已经开始下降，现在的桌面型装配机器人的价格已经接近于激光切割机的价格了。然而，该设备的成本不是你唯一需要考虑的因素。原因是，它们是为大批量制造而准备的，送入该设备的元器件是以**卷装料**（tape and reel）的形式提供的。这实际上就是一条很长的胶带，元器件在上面以一定的间隔分布，并且在胶带的一边（或两边）有一排孔洞，为的是让设备上的链轮可以带动胶带进给。然后这个胶带就被缠绕到一个卷轴上，每一卷上通常有几千个元器件。

很显然，对于电路设计中用到的每一种元器件，你都需要有一个卷轴。在任一时刻，可以被加载到拾取-贴装设备上的不同卷轴数量也是有限制的。在不同的卷轴之间切换既要花费时间，也会增加成本。

考虑到这些情况，如果正在使用拾取-贴装设备，最好再审视一下自己的设计。如果能让设计中用到的元器件种类更加合理（例如，也许一些电阻的阻值不是很关键，你可以用其他地方用到的阻值相近的电阻来代替），你就能减少最终可能成为库存、尚未用到的卷装料形式的元器件。

基于这种复杂性，所以最好把一些工作委派给封装厂（也称为合同制造商）去做。合同制造商是指做好各种准备，能帮助人们生产装配好的 PCB 成品的企业。通常它们能提供一系列的服务：从 PCB 设计，到与 PCB 制造商打交道，再到把元器件焊接到 PCB，甚至对成品电路板进行测试（我们很快就将对此介绍）。

使用封装厂就可以避免自己购买昂贵的设备。但把这些工作转移给其他擅长做这些事的人做，还有其他的好处。这些封装厂需要应对巨大的生产量，很自然会雇请专业人员来运行生产线。出乎意料的是，这些专业人员很可能在使用拾取-贴装设备时和在做任何的手工焊接工作

时都比你更加熟练，而且他们的人力成本也很可能比你低。这样就正好把你解放出来，让你有时间去做很多其他的工作，包括把产品推向市场，或者致力于你的下一个想法。

如果真的决定使用合同制造商，最好与他们讨论一下元器件的选择。对于常见的元器件，如果对其容差以及类似的参数没有特别的要求，也许可以明确说明，让封装厂使用现有的库存元器件，这样就不必为了使用少量元器件而购买整卷的此种元器件。

即便对于该合同制造商库存没有的、你需要购买的元器件，与该制造商合作，一同下订单，可能是合理的做法。如果该制造商与供应商在之前有过交易，它的声誉也许能让你达成一个更好的交易。

10.4.5 测试

现在，电路板已经全部制作和装配好了，但怎样才能知道它们都能按预期方式工作？这就需要用到测试。

实际上，在自动装配过程中，可能已经包含了一些测试步骤。装配线上可以包括自动光学检测（AOI）。在这个检测过程中，一个高分辨率的摄像头被用来对电路板及其上面的元器件的某些方面进行检查。例如，在电路板进入拾取-贴装设备之前，它能把获取的图像与一个已知的良品图像进行比较，检查焊膏是否已经被正确地涂敷。任何电路板，只要它与“黄金”参考版本的差别太大，就会被标记出来，以便让后续的熟练操作人员做进一步检查。

电路板通过AOI检查后，下一步就是对它们做一些功能测试。即使对于手工焊接的电路板，你也可以（并且也应该）做这一步测试。

最基本的功能测试就是给电路板上电，就像把它用在最终产品中那样，确保其行为符合预期。然而，这种测试方式可能会花费较多的时间。测试的重点不应该是确保实现所有正常的操作，而应确保以下方面：PCB及其上面的元器件已被正确地焊接；没有元器件存在故障；PCB本身没有任何的制造缺陷。

一个更好的办法是构建一个专用的测试架，用来对电路的不同部分做测试，并测量电路板上一些指定点的电压。然后把这些测量值和已知

的正确值做比较，自动做出一个决策，确定被测件（DUT）是否通过了测试。你可能会发现，在PCB上添加几个测试点（PCB上暴露在外并且连接到电路的有用部分的焊盘）会使得测试过程变得更加容易，因此，在完成你的PCB设计之前，值得考虑一下将来怎样执行测试过程。

构建一个这样的测试架不是太复杂，尤其是在你已经能使用像Arduino和树莓派之类的单板构建系统后。如果还做不到这一点，建议去（重新）看一下第5章。

如果不愿意花时间在每一个测试时都做几次单独的连接，也可以用测试架来测试。测试架的一般实现方式是利用PCB的安装孔对齐位置，然后用一些夹具让电路板与若干已经仔细调整好位置且装有弹簧的探针保持接触。这些探针被称为弹簧针。弹簧的使用意味着当把电路板放入测试架后，无需任何额外的工作，如进行焊接，这些探针也能与电路板保持良好的接触。

然后，利用测试程序进行测试，即在测试过程中的一些相关的时间点上，测试不同弹簧针上的电压，然后再检查被测电路板的运行情况。

如果被测件包含微处理器芯片在内，测试架能先烧录一个测试程序，用来帮助做测试；然后在测试结束时，假设被测件通过了测试，测试架甚至还能把最终的固件映像文件烧录进去，为其做好部署前的准备。

10.5 批量生产壳体和其他固定物

对于物联网装置的电子电路部分，我们已经介绍了怎样对其进行规模化的制造。但对于任何定制的外壳或用来构建最终产品的其他配件，又该怎样扩大其生产规模呢？

一条很好的用来压低生产成本的经验规则是，最小化一个人用在每件产品上的必要时间。机器往往要比人便宜，人工成本在总成本中占比越低，你就越有能力向参与到装置装配工作中的人们支付体面的薪水。

如果像第6章中讨论的那样，设计使用了一些较新的数字化制造技术，如激光切割或3D打印，在装配过程中，可能就已经没有多少需要人工完成的工作了。

不过，虽然最小化人工成本是一个合理的目标，但在各种与生产相关的参数中，它不是你唯一需要考虑的因素，生产速度也很重要。虽然3D打印机和激光切割机都相当地节省人力，但它们不是最快速的生产技术。如果你只需要一件物品，等几个小时，把它打印出来是可以的；但如果用这种方式生产一千件产品，则要么会花费很长时间，要么需要很多台3D打印机。

如果对所有可能的制造技术进行介绍，那足够写成一本书了。实际上，有很多书是介绍这方面的。我们特别推荐的一本全面介绍现有制造工艺的入门书籍：Rob Thompson所著的*Manufacturing Processes for Design Professionals*。

为了让你能对批量生产时所涉及的各种问题有所了解，我们来看看最常见的用于批量生产的制造方法：注塑成型法。

顾名思义，这种工艺方法主要是把熔融状态的塑料注入模具中以形成所希望的形状。在塑料充分冷却后，分离模具，用若干个顶针弹出工件，使其落入一个收集箱中。整个工作周期都是自动完成的，并且所花费的时间要比3D打印一个工件少很多。这就意味着，成千上万的部件很容易就能以较低的成本生产出来。

对于注塑成型，首先要制造模具，而这也是花钱最多的部分，被称为工具的准备工作（tooling up）。模具由钢或铝加工而成。必须要经过精心

的设计和抛光处理，以便成型过程能顺利进行，制成的工件具有所需的表面光洁度。模具表面的任何瑕疵都将被传导到用它们生产出来的每一个工件上，因此你应该确保模具没有问题。让工件的表面包含一些纹理能有助于掩盖任何的缺陷，而这也可能让工件的成品有更好的手感。通常对于超光滑的表面，需要采用一种被称为电火花加工（EDM）的工艺对模具做最后的处理。该工艺通过使用高电压火花，使金属表面气化，从而形成高光洁度的效果。

模具还需要包括容纳顶针的空间，以便把制作好的工件移走，以及一条让塑料流入模具的路径。如果你曾经组装过模型飞机或汽车，你应该熟悉这些路径。多余的浇口，使套件中的每一个部件保持连接的塑料支架（需要你把它掰掉），都是这些路径所遗留的。在装配好的产品，就会把这种浇口从部件上清除掉。

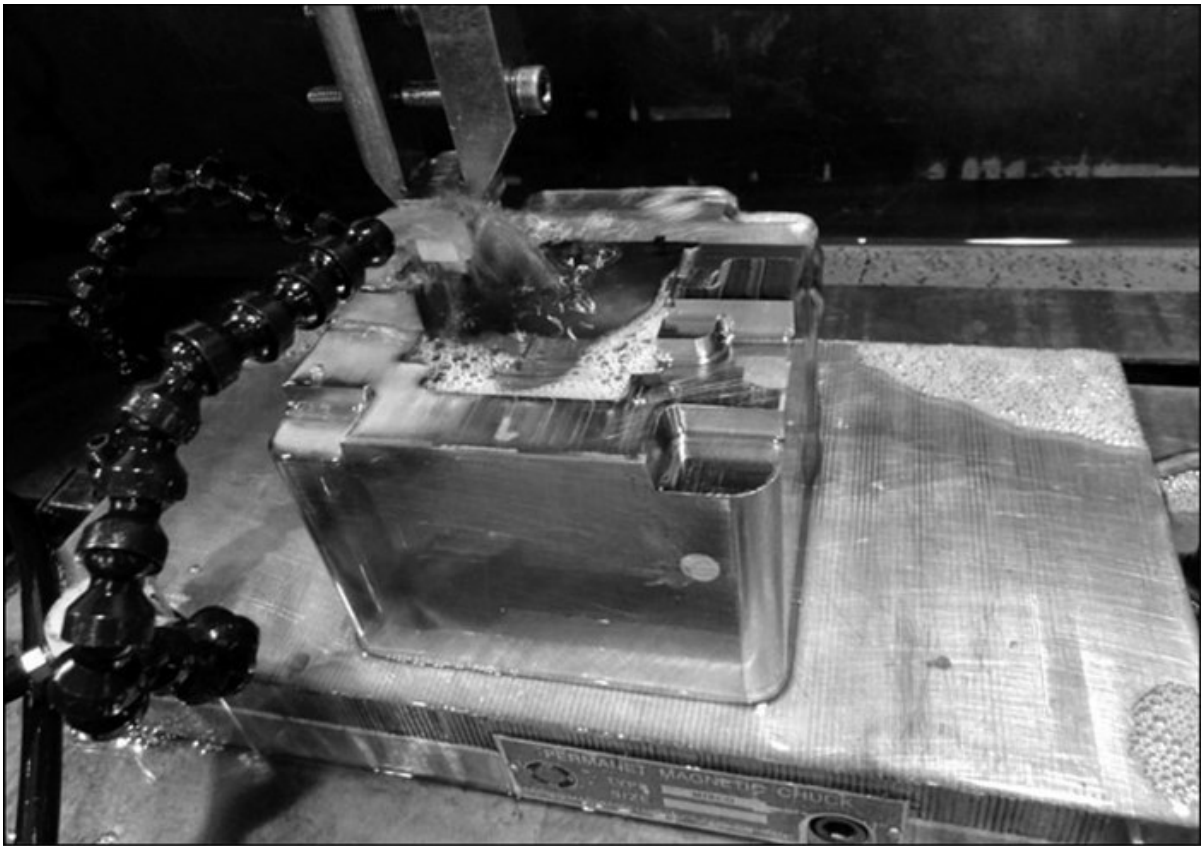


图 10-4 BERG的Little Printer打印机的一个模具在EDM机上做表面处理

和任何其他的生产工艺一样，注塑成型有其自身的设计注意事项。因为工件成型后需要从模具中取出，最好避免出现非常尖锐的拐角和完全垂直的表面。垂直方向上的一个很小的角度，即**脱模角度**（draft），为工件和模具的完全分离留出了余地，而一致的壁厚避免了工件在冷却后发生扭曲变形。

如果你想为了让工件更牢固而增加壁厚，那么不妨试试另一种替代方法：使用肋状结构（rib）增加工件的刚性，而无需额外使用很多塑料。看一下你已有的一些注塑制品的内部，你会看到一些用最少量材料获得最大强度的常见技术，也会看到用于固定PCB的安装点或用来把各个组件固定在一起的螺丝孔的成形方式。

最简单的模具被称为直拉模（straight-pull），由可以分为两半的两个模子构成。如果设计需要包含垂直表面或复杂的突出端，使用更复杂的从侧面引入额外部件的模具也是可行的，但会增加制作模具的成本。

减少模具制作成本并同时增加生产速度的一个办法是一次制作多个工件。如果你要的部件足够小，你可以在一个模具上复制很多个此类部件；或者，正如我们在模型飞机套件中看到的那样，把很多不同的部件收集在一起，制成一个模具。

在一种被称为**多组分注塑成型**（multishot moulding）的工艺中，你甚至可以让不同颜色的部件共享同一个模具。经过对每个部件所需的塑料量做仔细测量后，首先向模具注入一种颜色的塑料，把需要呈现为这种颜色的部件区域填满。然后，向模具注入另一种颜色的塑料，把模具的其余部分填满。显然，模具型腔中会存在一个混色区段，但经过精心的设计，该区段只是成为了浇道的一部分，因此可以被丢弃。

案例研究：BERG的Little Printer打印机

Little Printer打印机由伦敦的设计公司BERG制作。它是一台讨人喜爱的微型联网打印机。这个项目是一个很好的范例，从初始的概念到最终的产品，涵盖了物联网产品的整个研发过程（也正是本书的意旨）。

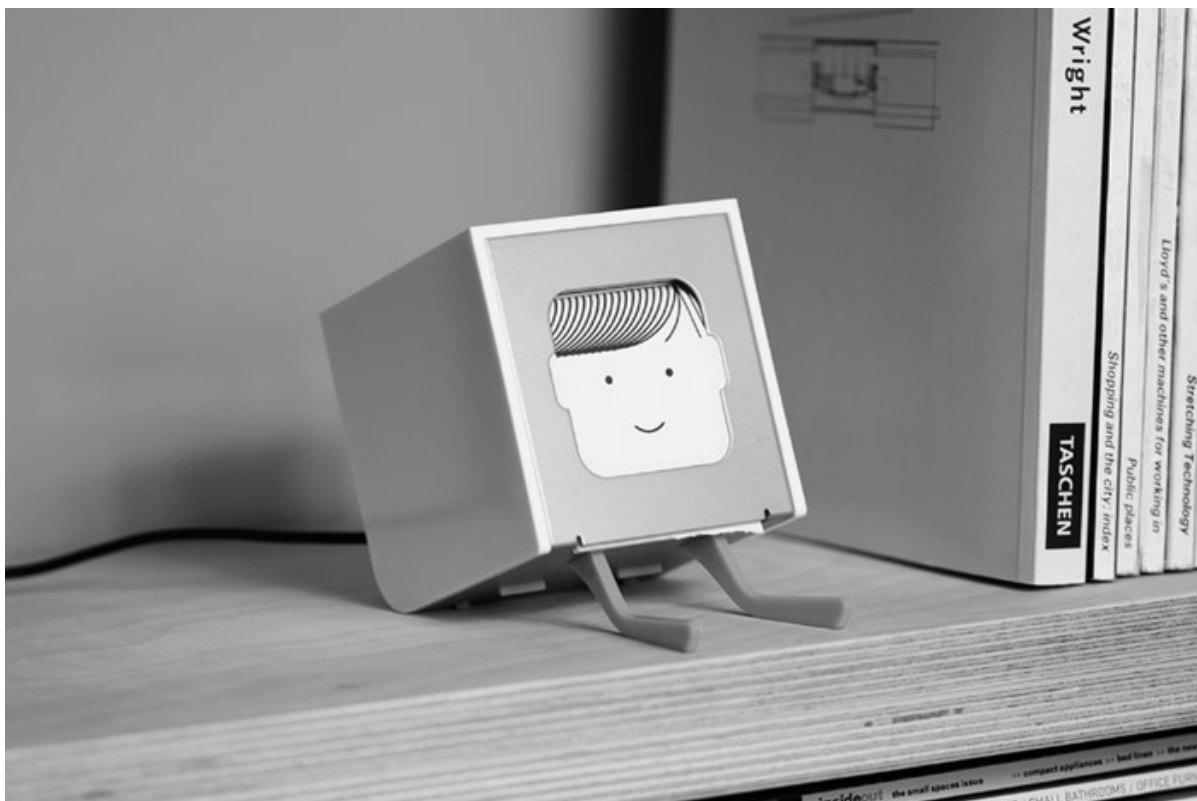


图 10-5 Little Printer打印机

为了能完整地了解这个过程，我们必须追溯到2006年。当时，BERG的创始人之一，Matt Webb，就怎样把连接在他电脑上的打印机共享给他的密友和家人（不管他们在世界上的什么地方），分享了一些他的想法（<http://berglondon.com/blog/2006/10/06/my-printer-my-social-letterbox/>）。他把这个设想的装置称为社交化的信箱。他的想法是，与他有密切交往的人将能够与他分享打印出来的内容，即使当他们在空间上远离他时。他还若有所思的写道，也许他还能订阅博客或类似的、不太个性化但仍然是亲自挑选的出版物，让它们在适合自己的时间传送过来。

顺便说一句，在对这个初始的博文的评论中，有一条来自于一个叫艾德里安的人的评论，他的建议是，比重复利用连接到你的电脑的打印机更好的方式是，让打印机具备自己的网络连接和智能。

这个最初的设想导致许多人，包括Matt和他的一些朋友们，以及更广泛的创客社区，尝试把改变了用途的收据打印机连接到Arduino

板，并接入因特网。他们尝试打印了天气预报、他们当天的预约活动，来自于朋友们的推文，等等。

从这些早期的调查研究中，他们逐渐解决了什么值得打印出来，什么不值得打印的问题。例如，推文由于数量太多，被证明不宜都打印出来。这让他们有机会探索收据打印的问题，更好地了解纸张的质量，以及相对于纯数字方式，为什么人们仍然对使用打印机存有偏好。

这些实验持续了数年，并且几乎一直是以一个后台项目的形式运作，因为BERG当时同时忙着做来自于客户的工作和自我导向的产品。虽然这些产品——**Avilabot**，实物形式的即时消息状态显示装置，和**SVK**，可用紫外光电筒查看隐藏文字的图形化小说——看上去与**Little Printer**没有直接关系，但在很多方面，它们使该公司对构建产品的过程有了更多的了解。

不过，为了把**Little Printer**变成实物，仍然有很多事情要做。

在结构形式方面，需要对打印机的机械结构进行精心的设计，并且为了制作外壳，需要同时为注塑成型和金属冲压两种工艺流程准备模具。项目团队在注塑成型工艺方面碰到了一些问题：塑料冷却的方式会导致在外壳表面上产生一些波纹，这主要出现在用来容纳按钮和状态指示灯的孔洞周围。尽管这个瑕疵很不明显，但却破坏了他们所追求的如钢琴烤漆般的视觉效果。解决这一问题的办法是，在制作外壳时，采用了两阶段的工艺流程。现在，在注塑成型时已经取消了孔洞，从而能得到正确的表面光洁度，然后再通过钻孔获得孔洞。当然这种工艺更为昂贵，但却能形成预期的更高质量的光洁表面。

在电子电路和软件方面，他们希望能把用户设置的工作量降到最小，并且希望从订购打印机到在家中使用它，用户都能对用户体验进行全程控制。于是他们就选择了基于**ZigBee**的无线方案。在与其他端点配对时，该方案不需要使用任何的用户名或密码。然而，虽然家用路由器支持**WiFi**（**802.11**标准），它们没有对**ZigBee**的内置支持。这决定了一个两盒解决方案：除了**Little Printer**打印机，用户还会收到一个BERG云桥（**Cloud Bridge**）。这个云桥就是一个包含了**ZigBee**无线模块和以太网端口的盒子。用户把自己所在网络的网线头和云桥的以太网端口相连，就能接入因特网了。

这使得该打印机可以与BERG的云服务器软件通信。在提供“出版物”递送服务的同时，服务器端软件也提供了面向用户的网站。通过使用网站提供的用户接口，BERG允许用户对他们的Little Printer打印机进行注册，以及选择他们想订阅的出版物并确定递送时间。

2011年11月，在最初的博文发表大约5年之后，这个项目的所有细节才浮出水面。

即便到了此时，仍有一些工作有待完成。到2012年8月时，事情接近完成，他们已经开始接受预订。此时，他们预测的发货日期是2012年10月中旬。然而，由于平衡所有这些不同的工作很困难，几个新的问题仍亟待解决。

开关电源的开关频率和云桥单元的PCB设计共同作用，导致了该装置没有通过电磁兼容性辐射测试。为了解决这个问题，供货时间被向后推延了一个多月。然后，在外壳上安装磁铁夹时的一个小问题，又让供货时间延迟了一个星期左右。

到2012年11月底的时候，这些有趣的Little Printer打印机开始被迅速发送给客户，让众多家庭接触了物联网的美妙之处。

10.6 认证

创建物联网产品的一个不太明显的方面是认证问题。如果你忘了制作PCB，或者，为你的装置所编写的软件只完成了一半，它不能按预期方式工作，事情很明显没有做完。不符合相关的认证或规定，产品同样是不完备的。不过，只有把它发送给经销商，或在更糟的情况下，当它已发售时，你可能才意识到这一点。

总体来说，这些规定的存在是有充分理由的。它们能让那些日常使用的产品在安全方面更有保障；确保这些产品能与来自于其他供应商的配套产品一同正常工作；确保某个产品不会产生大量不必要的电磁辐射，从而避免对附近其他装置的正确操作造成干扰。

你可能以前没有注意到，如果仔细看一下手头的任何小型电子装置，你将会在它的某个位置发现一组标志，如CE、FCC、UL等。每一个这样的标志都代表一组特定的规定和测试，同时也表明该装置已经通过了相关测试。例如，CE标志表示符合了欧洲标准，FCC表示遵守了美国联邦通信委员会的规定，而UL则表示通过了独立评测实验室UL的测试。

装置需要通过哪些认证，取决于它的确切功能、目标市场（消费者，工业等），以及预期的发售国。协商完成所有这些事项劳心费力，所以最好是与一家本地的测试机构合作。他们不仅能帮你做测试，而且也能为你提供建议，例如，装置需要遵守哪些规定，不同国家间的规定有怎样的差别，等等。

这样的测试机构会对装置进行大量的测试，它们远远超过装置在正常使用时会遇到的情况。它们有可能包括：检查材料的规格参数，以确保装置没有使用含铅涂料；给装置施加8千伏的静电冲击，看它如何应对；用加热到500摄氏度的发热丝触探被测装置，检查它是否会被引燃；等等。

特别令人感兴趣的是电磁兼容性（EMC）测试。它既对你的装置易受干扰的程度做测试（干扰可能来自于其他电子装置，或者输电干线上的浪涌，等等），也会测试产品自身所产生的电磁干扰。

电磁干扰是电路电流发生改变时生成的“电气噪声”。当人们有意利用这种电磁干扰时，可能会很有用处：无线电和电视广播就是利用这个现象远距离传输信号的；移动网络和其他的无线电通信系统也是如此，如WiFi和ZigBee。但电路无意中发出足够强的信号时，就会扰乱所需的无线电频率，因而就会出现问题。有时你会注意到，当手机开始振铃前，附近的一个在隔离电磁干扰方面做得不太好的立体声音响发出“嘀，嘀-嘀-嘀”的响声。

被测件需要符合整个产品的最终规格。所有的测试都是用它来完成的。因此，测试将最有可能成为产品交付时间表中的关键点。测试中发现的任何问题都将导致发货时间的推迟。在此期间，你会通过一次新的版本迭代解决这些问题。

进行EMC测试时，为了尽可能地减小外部电磁干扰对测试的影响，装置会被放入一个与外界隔离的电波暗室。然后让装置正常运行，与此同时，在距离被测件3米远的位置用频谱分析仪监测任何的辐射发射情况。该测试给出了测试规程中指明的各个不同频率上的射频辐射水平。如果在任何一个频率上的测量值很接近限制值，就重做一次测试，并且改为在距离10米远的位置监测频谱。在这个较远的距离，可接受的限制值也较低，而这正好能检查信号的衰减速度有多快。运气好的话，10米远处的测量值仍在限制值之内，你将能获得认证。

所得到的测试报告将被添加到技术文件中，并将被符合标准声明所引用。编写测试报告是认证的需要，同时测试报告也是对设备的关键信息以及它所经过的测试的一种记录。

除了测试报告，你还需要把PCB设计文件、加工装配证明书、你所使用的任何预先认证过的模块的认证证书，以及重要部件的数据手册都收集起来。制造商（也就是你）需要把这些信息都保存到一个安全的地方，以便在有关部门需要查验时能及时提供。

技术文件所在的位置会在符合标准声明中提及。在该声明中，你需要公开宣布你的装置符合各种规定中的哪些指令条款。

对于某些规定，你还必须通知某个特定的指定机构。例如，如果电路是用作无线电通信，因而会有意地发射电磁干扰信号，当它要在美国

销售时，就必须要到FCC注册。这种注册是除了通过发布符合标准声明做自我验证之外，需要制造商额外完成的工作。

另外，由于一些更为复杂的指令型条款（有意发射器规则就是一个最好的例子）会增加复杂性和开销（包括管理和财务两方面），使用预先已经核准的模块往往比较明智。

因此，装置中可以包含预先核准的模块，如WiFi模块（芯片、天线和相关电路），或电源适配器，并且不必重新做所有相关测试。只要不以任何方式修改这个模块，该模块的制造商所做的认证就被认为是充分而有效的，因而就只需要把相关的认证文档囊括到技术文件中就可以了。

在欧洲，你还必须进行废弃电子电气设备指令（WEEE指令）注册。它并不涉及产品的技术层面，而是以减少需做填埋处理的电子废物的数量作为目标。欧盟各国已经建立了面向电子电气产品的生产商和零售商的回收计划，以鼓励对这些装置进行更多回收，并对回收所产生的成本予以资助。

零售商既可以自己运营一个回收系统，来接收人们不需要的电子装置，也可以加入一个回收计划。该计划的运营者能代表零售商们负责回收工作，但通常会收取会员费。

在英国，环境署对生产商可以加入的回收方案列表进行管理。一些方案的提供者会根据公司规模或其生产的电子电气设备的数量，把会员分为不同的级别。对于规模较小的生产商，例如那些生产的电子设备不到一吨重的生产商，采用每年固定收取几百英镑的方案。较大的生产商则需要定期（通常是每季度）报告发货总重量，并依据这个重量按比例付费。

10.7 成本

正如我们在本章的其余部分所看到的，进入大批量制造阶段后，你需要考虑很多事情。但其中很多事情都涉及庞大的前期成本。实际上，你涉入制造过程的程度越深，对核心的编码技术或关键的设计技术的需求就越少，而花在平衡现金流和筹资上的时间就越多。

不过，做这些事情是有好处的。你将能让更多人得到一款很棒的联网装置，而这太重要了。

如果已经预先募集到了足够的资金，管理“制造项目”的任务会比较容易。但与所有的项目一样，你还需要照看一下工作时间和关联事物。如果你以前没有经历过这些，特别是如果你在管理相互关联的任务的交付和截止日期方面没有太多经验，就值得向一名值得信赖的顾问或合作伙伴寻求帮助。

不需要会运用甘特图或有PRINCE2项目管理资质，但团队中的某位成员需要密切关注事情进展，并且要决定在继续推进其他任务之前，哪些事情必须先做。另外，还要关注项目时间线上将来发生的事，在问题成为团队其他成员的障碍之前，发现并处理它们。

限于本书篇幅，我们不会在项目管理方面提供一个完整的入门介绍，但可以让你对需要留心的各种事情有一些概念，并让你了解一些有助于估算成本的经验规则。

随着全球供应链的持续优化，作为消费者，我们已经变得习惯于让我们的需求立即得到满足。即便是订购实物形式的物品，我们也常常习惯了在下单之后次日即能收货。

然而，批量制造的优化目标仍然是产品的产量，而不是做准备工作的速度。因此，从订购一件物品到该物品能被使用，所需时间之长有可能会让你感到震惊。

你很有可能会在制作PCB时，最先遇到这种情况。从把Gerber文件发送出去，到得到焊接好的电路板，常常会花费数周时间。更快的周转

时间不是没有可能，但往往要支付额外的费用。

对于需要进行设置和准备模具的工艺流程，如注塑成型，情况会更糟。模具的制作、测试和微调可能会花费一两个月的时间。在这个过程中，有时你很忙碌，但在大部分的时间段里，你只能坐等那些工作结束。

通过精心的计划，你将能够在上述的等待时间里，继续做项目其他部分的工作，例如，可以完成服务器端代码的编写，或者准备营销方面的材料。但在项目计划中，仍然需要留出这些做准备工作的时间。

搞清楚项目的大概成本需要一些技巧。显然，这在很大程度上取决于试图达成的目标的复杂度。

很多在线的PCB服务都包括了一个报价工具。因此，甚至在设计完成之前，你就能估算出大概的价格。对于所需的PCB的尺寸和它可能需要的层数，你应该能做一个合理的估计，而这些正是报价工具用来计算费用的主要因素。如果想要一些与众不同的东西，如不同颜色的阻焊层，这些额外的因素也要考虑在内。

对于装配PCB，你应该留出大约150到800英镑的预算，用作生产准备的成本，例如，制作焊膏模板，为拾取-贴装设备编程，等等。每个元器件的装配成本在3到14便士之间。元器件本身的成本没有包括在内。

当小批量购买元器件时，你经常会注意到，价格列表也会显示购买更多数量时的价格。这是一个好的开端，但通常这些列表不会显示购买数量大于1000的价格。

假设你可以找到一个购买数量为几百的报价，你可以使用Archard法则推算出一个更大购买数量的指导价格。Archard法则是Lawrence Archard提出的，是这位硬件工程师半开玩笑的说法。和本书的两位作者相比，他在制造方面所拥有的经验要丰富得多。他认为，当知道了购买几百个元器件的价格后，每当购买数量增加10倍时，单件成本就会下降到原来的四分之三。

在很大程度上，塑料部件或其他结构件的成本取决于它们的设计。我们很难在这里给出有意义的数字。不过，由硬化钢制成的模具在需要

更换之前，能生产出10万个部件，但其制作成本轻易就能达到1万英镑。

为了能通过认证，你可能需要再花这么多钱。对于较简单的装置，需要被认证的领域较少，通过认证可能要花费大约2000英镑。对于更复杂的设计，特别是那些包含未经认证的射频模块的装置，完成所有认证的成本可能是前者的10倍（2万英镑）。

当然，随着项目的进展，当你直接与供应商交流，准备向他们下订单时，你将会得到更准确的报价（并最终获得完全准确的收据）。你可以据此更新物料清单和成本电子表单，并且把这些新的信息应用到你的计划中。

10.8 扩展软件

事实证明，对于一个物理装置，制作一个它的原型和制造一件它的产品，是两件完全不同的事情。和最终被摆上货架的产品相比，初始的原型很可能具有不同的尺寸、形状、颜色、材料、光洁度和质量。然而，软件是无形的和易改变的。它既没有需要订购的部件，也没有需要支付的物料清单。运行在装置中或因特网上的程序是由比特信息构成的，而它们是不可见的。在项目开发阶段编写的软件和最终运行在产品中的软件是无法用肉眼区分的。

然而，与装置的外形结构和电子电路单元一样，在实际发布软件之前，也必须要对它进行修改和完善。我们将首先介绍在嵌入式装置上以及已经构建好的任何在线服务环境中部署软件时所涉及的事项，之后再看一下需要修改和完善的各种要素，即正确性、可维护性、安全性、性能以及社区。

我们在这里只是对这些事项做一个简单的介绍，为的是让你对所涉及的问题有所了解。对于如何构建和部署安全、可靠的Web服务的方式来说，你所选用的特定语言和框架的相关资源一般都会有更详细的介绍。

10.8.1 部署

把软件从开发机（笔记本电脑或开发团队的源代码库）复制到产品中将来运行它的位置，这一过程通常被称为部署。或者，有时候会用实物产品的说法将之称为**出货**（shipping）。

就运行在微控制器上的固件而言，在装置的制造过程中，部署工作（有希望）只做一次。软件将被烧录到装置中，其所采用的方法跟在开发阶段更新原型装置所用的方法比较类似。对于象Arduino这样的通常只能运行单一程序的简单装置来说，这一过程与开发阶段完全一样。对于BeagleBone或树莓派之类能运行完整操作系统的装置，你会愿意以一种便于你快速、可靠地制作新构建版本的方式，把各种程序代码、库文件和用到的其他文件“打包”。也就是说，用一条命令，把所有这些文件安装到一个新单板上。

用于在线服务的软件往往会在单一的位置上运行。然而，它通常在整个因特网的范围内都是可见的。这意味着它有可能受到较大负荷和一系列意外或恶意输入的影响，从而导致无法正常运行。另外，这些代码往往会比较复杂，并且构建在较多库代码的基础之上，而较大的复杂度意味着出现错误的可能性也较大。最后，作为一个易改变的面向用户的软件产品，总是有可能对代码做更新，通过引入新功能或改进设计来增加价值。

这一切导致的结果是，与装置中的软件相比，对在线软件组件做更为经常性的更新，不仅是可行的，也是必要的。拥有一个好的部署过程能让顺利安全地完成部署。在理想情况下，只需一个触发动作（如运行一个脚本，或把代码推送到代码库的一个发布分支），一系列动作就会随之发生，用来对服务器上的软件进行更新。如果使用的是托管服务，如Heroku，你会有简单、标准的方法来做这件事情。如果运行的是自己专用的Web服务器，或者可能运行的是一个虚拟机，如一个Amazon EC2实例，从使用scp、rsync或git复制代码的shell脚本，到诸如Capistrano之类的部署框架，也有很多解决方案可选。在不久的将来，一个有吸引力的选择是Docker.io。它允许你把在笔记本电脑上运行的应用程序打包为一个虚拟的“容器”，而这个容器能在一个面向因特网的服务器上运行且无需做任何的改变。

10.8.2 正确性和可维护性

假设你已经出售了一千台联网咖啡机。恭喜你！现在该做祈祷了，祈祷它们能够正常工作，而不至于让你收到一千份投诉。错误有可能各种各样，也许当客户用它做拿铁时，它却做成了卡布其诺；也许咖啡还没煮好，它就发推文说“咖啡已经煮好了”，或者情况正好相反。

显然，作为一个大众产品，软件必须正常运行，并且要高效、安全地运行。

在部署软件之前，为避免上述情形的发生，一项重要的步骤就是测试代码。你选择的编程语言和开发框架会有标准的、易于理解的测试环境，能用来帮助你降低测试工作的难度。

由于服务器软件处于一个中心位置，无论是修复错误或是引入新功能，对其升级都很容易。对于Web应用，这是一个真正的好处，因为它消除了一整类的支持问题。

然而，对于装置中的嵌入式代码，对其做测试尤其重要。因为，产品一旦被发送到用户手中，对其升级就很困难了。考虑到该装置不管怎样都能接入因特网，所以采用OTA（空中下载）技术更新代码是可行的，而这正是某些产品（如Electric Imp平台）的卖点之一。不过，使用OTA时要谨慎：如果更新固件造成了家庭供暖系统的故障，那该怎么办？此外，随着物联网产品正越来越多地融入我们的生活和家庭，它们已成为一个对骇客颇具吸引力的目标。如果你的装置能够更新代码，则必须保证用来传送和验证任何更新内容的通道必须稳固可靠，以免骇客加以破坏和利用。正如移动技术专家Brian Proffitt警告的那样，“物联网可能会试图杀死你”（<http://readwrite.com/2013/09/18/internet-of-things-security-disaster-terrorism-war>）。

10.8.3 安全

这些关切使我们再次回到重要的安全问题上。虽然我们在讨论软件的原型设计时强调过这一点，但考虑一下大众市场产品遭受安全攻击的后果，却是至关紧要的一件事。

我们已经讨论过使用HTTPS的加密方式。是否需要使用HTTPS可能会取决于对原型平台或产品平台的选择。除了要确保客户端装置和Web服务器之间通信的安全，你也应该考虑一下服务器自身的安全。作为一个始终保持运行、始终联网并且对整个因特网可见的设备，服务器是最明显的攻击目标。这些注意事项并非专门针对物联网的，但系统管理和开发团队应该对此熟知。以下是一些比较重要的原则。

- 确保服务器已经安装了最新的安全补丁，已经处于适当的防火墙软件的保护之下，能检测密码破解攻击和rootkit攻击并能减轻其影响。
- 永远不要以明文的形式存储用户密码。如果数据库的安全性受到破坏，攻击者可以轻易地以任何用户的身份登录。正如我们在第2

章关于“散列”的补充资料中所谈到的那样，应该用一个安全的、目前尚不能被轻易破解的算法对密码加密，并且应该通过“加盐处理”获得额外的安全性。

- 千万不要轻信用户输入。对输入Web应用的任何内容都做检查，看它是否符合你所期望的数据类型。拒绝或清理掉任何不相符合的内容。虽然你可能会觉得来自于你的物联网装置的输入不会有问题（因为其代码是你编写的），但有可能它已遭到入侵。或者，攻击者可能会冒充它。特别要提防的是，不要把未经检查的用户输入传入数据库（否则，如果其中包含了SQL命令，你将面临SQL注入攻击的风险），或者，不要把未经过滤的用户输入包括到你的HTML页面中，因为这可能会允许跨站点脚本（XSS）攻击。去除所有的HTML标签（或只允许有限数量的用于格式化的标签）或对输出进行转义处理。
- 要注意来自于其他恶意的或已被入侵的网站的跨站请求伪造（CSRF）攻击行为。例如，如果你的一位用户浏览了一个有问题的网站，并且该网站的网页使用JavaScript打开了<http://some.example.com/heating?switch=off> 这个指向你的网站的链接，而他此时已经登录了你的网站，那么，他可能回到家后会发现屋里好冷。

10.8.4 性能

当考虑软件的扩展时，很多人想到的第一件事是：它是否足够快，是否能应对大量的用户。但实际上，我们已经介绍过的其他一些因素通常更为重要。如果Web服务运行在一个现代的框架之上，通过把程序部署到一台性能更强的机器上，或者通过运行多台服务器并使用一台前端服务器或代理服务器管理负荷，就能轻松地实现扩展。一般情况下，应该专注于在适当的、足以应对大多数问题的、标准基础设施之上运行Web应用。如果非常幸运地获得了如此大的流量，以至于需要扩展软件，应该总是使用如下的算法进行优化。

- (1) 确认你确实碰到了一个问题。
- (2) 对运行缓慢的任务进行测量和分析，找出问题所在。

(3) 解决这个问题。在分享如何解决此类问题的最佳实践方面，网络社区做得很好。因此，你经常会发现其他人之前已经碰到并解决了此类问题，并且记录下了他们经过实际验证的解决方案。

10.8.5 用户社区

无论你是推出面向大众市场的商业产品，还是面向开源社区发布成果，只有人们真正使用，你的项目才能成功。虽然少数几家大公司能拥有苹果公司著名的承诺——“近乎疯狂般优异的客户服务”，小型、专注的创业公司常常只能在响应能力和热情方面与它们匹敌。你至少需要有一个客户支持电子邮件地址或一个bug报告工具（最好公开可见，这样用户就能很容易地找到相关的问题），你也可以充分利用各种社交媒体论坛上，如Twitter、Facebook等。然而，除了响应各种询问，还应该在发布产品前考虑一下用户体验：网站上是否有文档、介绍性的视频和教程？定期地就产品开发和相关话题发表一些博文，有助于建立一种由当前用户及潜在用户构成的读者群。最后，利用某种形式的论坛、邮件列表或聊天室，也能让用户之间相互支持，从而减少你的工作量。更重要的是，它们有助于围绕你的产品，建立起相关社区，逐步积累起相关的专业知识。

10.9 小结

我们在这一章中，填鸭式地介绍了很多的实例、工艺流程和建议。希望你现在没有被弄得晕头转向。

构建实物形式的产品很困难，而构建能够连接因特网的产品则变得更为复杂。然而，最终成品产生的反响是网站或移动应用所不能比拟的。

制造阶段必然会涉及很多技能和任务，而它们与你用来实现想法的技能和工作的有着很大不同。当你了解了所有要做的事情后，可能会有很强的畏惧感。

然而，这一过程在这个世界上，已经经历了一百多年的完善。即便在电子行业，这个过程也有几十年的历史了。因此，如果能找到正确的合作伙伴，它就没有你想象的那么困难了。

第 11 章 道德伦理

人们可以容易地以一种非黑即白的方式看待任何的技术。也许你可能认为，技术会：

- 消灭工作岗位；
- 具有入侵性，会让我们受其控制；
- 割断人类与古老传统间的联系；
- 鼓励我们变得懒惰和不健康；
- 诱使我们认为自己像神一样无所不能。

或者认为技术：

- 是新鲜亮泽的；
- 是一种进步（因此是在暗示它更好）；
- 将拯救生命和让饥饿的人填饱肚子；
- 将（至少）能使我们更快乐、更健康、更安全，接受更好的教育；
- 将我们从单调乏味的工作中解放出来，以便能有更多的时间用于休闲；
- 将创造更有趣和更值得去做的工作，以代替那些被它淘汰的单调乏味的苦差事。

我们注意到，很多乐观的表述要比悲观的表述更复杂。其原因可能是，拒绝新的、不同的事物已深深地植根于这个社会，因而可以用更为发自肺腑的措辞进行表达。

花点时间回想一下你对技术的总体立场。你有可能总体上是反技术的或亲技术的（如果你正在阅读本书，也许后者的可能性更大一些）。你还有可能乐观看待某些的技术（因特网、太空旅行等），而对其他一些技术（转基因食品、风能发电等）持悲观立场。

由技术引发的情绪可能是强烈的。这让人想到了两个对立的与“美好生活”的哲学理想背道而驰或相向而行的华丽表述。

- **人类从一个较好的状态螺旋式地下降**：这可以被视为与宗教观念等同，如基督教的“人之堕落”，或简单地说，与传统的价值观等同。
- **人类通过技术走向完全自我实现过程中的一个不可阻挡的决定性进步**：这可能会导致该物种进入一个新的状态（后人类奇点或人类扩张至其他星球），或者干脆可以认为，每一个进步将引领我们进入一个稳定的乌托邦。

虽然这些表述是强有力的，但对技术专家来说，更为有用的是能够分辨更为精细的灰度级别。也就是说，是否要说服他人或质疑自己的信仰，以便在一个更为广阔的背景中重构它们。

在许多传统的学科领域，有一些道德伦理方面的课程，例如，工程或计算机科学类的大学课程中就包含了相关的课程。正如开放数据和创意空间专家Laura James所指出的（在OpenIoT的讨论中，本章后续部分还有详细介绍），“没有经历正规工程训练的骇客会错过学习这些课程的机会，这是不是一个问题”。更何况，正如我们在第1章的概述中所看到的，从相关的技术领域，到设计，再到艺术，物联网的从业者可能具有不同的背景。

在本章中，我们没有对道德伦理做正规的完整介绍（请感兴趣的读者参阅Mike W. Martin和Roland Schinzinger所著的经典教科书《工程伦理学》），而是对它的几个方面做了介绍。这些方面中的大多数是和正规的道德伦理课程的教学大纲有关联的。我们会从物联网相关的角度考虑它们。我们将介绍很多极端的、具有挑战性的想法。其中的一些想法可能看起来像是科幻小说。正如Martin和Schinzinger在他们的书的引言中写到的：

我们会赞同特定的立场观点争论，因为这种做法能更好地服务于我们的目标，即：鼓励批判性的判断，而不只是摘录别人的观点。因此，我们的目的不是要把什么观点强加于人，而是要引发人们对我们观点的理性接受或拒绝。

——《工程伦理学》

11.1 描述物联网的特征

让我们首先总结一下物联网是什么，从而对它能给人类与“美好生活”之间的关系带来什么特别改变这个问题有所把握。

我们已经强调了一个事实，即某种强大的技术（计算机芯片）突然变得廉价和丰富了。这绝不是一个新发现，并且该技术也不是第一种经历这一爆炸性转变的技术。颇有影响力的技术专家和决策者万尼瓦尔·布什（Vannevar Bush）在1945年发表的经典论文“诚如所思”（As We May Think）中这样叙述道：

如今，人们能十分轻松地造出由可互换零件构成的机器。尽管构造十分复杂，但它们却能可靠地工作。不起眼的打印机、电影摄影机或汽车都能证明这一点。当你能完全了解电触点的特性后，就会知道它们并不那么可靠。但注意一下自动电话交换机。尽管它有着数以十万计的此类触点，但仍然能可靠工作。一个金属网被封在一层薄薄的玻璃壳中，给一根金属丝加热，使它达到炽热发光状态，简单地说，这就是无线电装置上的电子管。人们已经制造了上亿只这种电子管，把它完全封装好，插入管座——它就能用了！它在制造时涉及的金属细丝的精确定位和调整可能会占用专业技师数月的时间，而现在制造它的成本是三十美分。这个世界已经进入了一个能让复杂设备兼具高可靠性和低价的时代，而某种事物注定要从中诞生了。

——www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/3881

技术可获得性的提高，使得某些能力不再只是为重要人物所拥有，普通公民也能触手可及。布什列举的那几个早期的例子涉及出版、交通和通信。物联网领域的进展也主要是和通信相关，但现今的不同之处在于，这一切无需技术专家或政治精英的专业知识或许可。从社交关系（有没有人在推特上提到我），到环境分析（利物浦的一个典型的住宅会消耗多少能源；福岛核事故发生后，日本各地的辐射水平是多少），我们可以发布和传播海量的数据。确实是“某种事物注定要从中诞生了”。

在第1章中，我们也指出过，“形式服从于功能”这句格言主要适用于物品的物理用途、它的功能可供性、传感器和执行器，并且只是最低限度地适用于数字通信。这就导致了某些物品看上去平淡无奇，但却具备任意的、可能让人意想不到的能力。

把因特网连接到真实世界，既使得你的物理行为得以公开，也在相反的方向上，使得因特网上的事件能够影响你的环境。把这一双向通信的过程应用于物联网装置，就会引起与隐私这个概念的交互。当你打开你的晚安灯时，在你母亲床头的小灯也将点亮，从而让她知道你在家里。当你离开办公室时，家中的WhereDial会指向适当的位置，从而让你的伴侣知道你在路上了。

我们已经多次指出，物联网的构成是：

物品 + 控制器、传感器和执行器 + 因特网服务

上述每一个方面，在与物联网具体相关的道德伦理问题中，都扮演一定的角色。我们在后续章节中会提到它们。

11.2 隐私

作为一个大规模的开放式发布平台，在隐私方面，因特网已经成为了一种破坏性的力量。你写的任何东西都有可能被网络上的任何人看到：从你早餐吃的什么这样的琐事，到你发布的工作相关的博文；从与你的兴趣爱好相关的文章到关于你发布的和朋友聚会的Facebook帖文。公开这些数据是有价值的：你在因特网上讲述的故事成为了你的面具人格，描绘了你在朋友、家人、同事和潜在雇主心目中的形象。不过，你总是希望人们能够看到这些数据吗？随着存储能力的迅猛增大，对这些数据的存储和搜索都很容易实现。你希望除了你的家人和朋友，公司、政府和警察也永远能够看到你的信息？一个常见的说法是：“如果你没有什么可隐瞒的事情，那你有什么好怕的。”这个说法有一定道理，但它忽略了某些重要的细节。其中一些细节可能不适用于你，但适用于其他人：

- 你可能不想让你的数据被一个会恶语相向的前配偶看到；
- 你可能有被罪犯、恐怖分子刺杀的风险；
- 你可能属于一个群体，并且因为自己的身份（宗教、性别、政党、新闻工作者）而被人盯上了。

更现实的情况是，你做出改变并重塑了自己的形象。然而，你过去的不良行为（醉酒的照片、政治声明）在将来可能会被别人用来对付你。

让我们现在看一下物联网是如何与隐私发生相互作用的。物联网与物品有关；与计算机相比，这些物品根植于不同的环境；物联网使得数据上传更加无处不在。我们看一下手机，特别是能连接因特网且带摄像头的手机。虽然我们通常不把手机看作是物联网装置，但带摄像头的手机拍照是物联网装置最典型的任务。在过去，你必须经历拍照、冲洗、把洗好的照片拿到计算机跟前、扫描和上传等一系列步骤（或把你的数码相机拿到计算机跟前，经由USB接口传输照片）。而现在，你可以在醉酒状态下，一键上传有失体面的照片。做某件事情

的能力存在于某个限定的环境（个体）中，而不是受一套包含多个步骤、最终止于通用计算机的流程的束缚。

即使是无伤大雅的照片也能泄漏数据。随着GPS坐标（很多相机和绝大多数智能手机都能生成）被嵌入照片的EXIF元数据，通过分析你的Flickr/Twitpic/Instagram图片提要，攻击者就能很容易地推断出你家在哪里，在哪里工作，甚至你的小孩在哪里上学。即便你剥离了这些数据，相片处理技术也允许对相似的照片进行搜索，而这些搜索出来的照片可能包含了这些坐标或其他线索。

对于运动跟踪数据，无论它是由实际的物品（如Nike+或GPS手表）生成的，还是由虚拟物品（如智能手机上的RunKeeper应用程序）生成的，都存在类似的问题。这些数据对于跟踪进展情况非常有用，而与朋友们分享你的跑步路线图、速度、心率等信息可能会有激励作用。但类似于上述情况，攻击者可以很容易地推断出你家的位置（可能离你开始和结束跑步的地方不远）并且获知你一天当中很可能不在家的时间。

当我们告诉家人和朋友关于晚安灯或WhereDial的情况时，他们常常会不太高兴，并开始嘟嘟囔囔的说一些隐私受到侵害之类的话。人们知道你在哪里，这个念头能引发强烈的情绪。然而，知道你的亲人是安全的，这个想法同样也是根深蒂固的人类情感。如果你允许你的位置信息被你所选择的人分享，就不存在侵犯隐私的问题。不过，几个月后，当你深更半夜回到家时，你会发觉当初给你母亲一个晚安灯的决定似乎不太明智；或者，你可能会后悔给你的另一半一个WhereDial，因为她可能会开始猜疑你的一些行为是否清白。

即便这些装置的设定本身是尊重个人隐私的，但攻击者仍会利用安全漏洞或安全机制的缺乏而获取信息。例如，如果读取一个从goodnightlamp.com的服务器到一个家庭的IP分组是可能的，你能查明相关的“大灯”是否已经关闭吗？即使该数据分组是加密的，攻击者不能根据一个分组被发送出来这个事实，推断出什么信息？（即，服务器是不是必须周期性地发送加密的“无状况”分组？）负责的物联网装置的制作需要非常审慎地考虑这些风险。在第7章和第10章，就怎样确保产品在线组件的安全性，我们讨论了处理技术的细节。读者可以再回顾一下这些内容。

到目前为止，我们看过的这些装置是你自己选择部署的。但由于传感器数据是如此的无处不在，除了你选择公开的数据，传感器会不可避免地检测到更多数据。

首先，我们在之前已经看到，很多“物品”如果只看外表，看不出它们连接到了因特网。当你从衣帽架上抓起一条能连接因特网的围巾，或者坐在一把能连接因特网的椅子上时，你能发现任何明显的迹象，表明有数据将被传输或某个行为将被触发吗？根据城市规划专家和技术专家Adam Greenfield的记载，带有隐蔽摄像头的交互式广告牌能对看广告的人的统计数据记录，而自动售货机也是采用类似的技术选择被陈列的产品。他对此做了评论，认为除了入侵公共空间之外，这些物品不只具有它们看上去具备的功能，而是能“预测行为并鼓励规范的行为”，并且这一功能是不透明的（<http://storify.com/clarered/adam-greenfield-connected-things-and-civic-respons>）。

反过来说，用数字化技术实现的物品也许不会像其对应的传统类似物品那样能给出细微的提示。比如，与纸质书相比，对一部电子书进行快速浏览要更困难一些，因为电子书没有来自于纸质页面的触摸反馈。很多早期的具备拍照功能的手机都增加了相机快门的模拟声，只是因为社会上对利用手机进行无声偷拍的担忧。

此外，让我们再看一下智能电表。对用电量进行实时精确的测量，有很多值得赞赏的目的。理解使用模式也能有助于电力公司选择正确的时间发电，从而避免过量发电，并能优化效率。在化石燃料资源变得稀缺，而且使用后的负面后果越来越严重的时代里，随着人类消耗的能源越来越多，能源的环保和可持续性变得越发重要了。电力公司收集的聚合数据对高尚的环保目标是有用的，但其中的个人数据会被怎样处理？

如果能对这些数据进行挖掘，把伴随电茶壶或咖啡壶进入加热状态一同出现的不明显的电量峰值找出来，也许能推断出某户人家在看什么电视节目。如果在早晨出现了四个较长的峰值区间，这可能表明，有四名家庭成员起床，并且他们在上学或上班前，用电热淋浴器洗澡。现在，如果你把这些数据与一些其他的数据，如水表的读数，综合起来进行分析，会是怎样的结果？智能电表当前正在欧洲范围内做推广，并且很快会在事实上变成强制性推行。欧洲数据保护主管助理

Giovanni Buttarelli曾警告说：“与其他来源的数据一起，对数据进行广泛挖掘的潜力是巨大的。”（www.guardian.co.uk/environment/2012/jul/01/household-energy-trackers-threat-privacy）

分析多个巨大的数据集的想法现在已经成为现实。做这些分析所需的智能算法和计算能力都已具备。通过结合长尾的两端（一端是廉价和无处不在的物联网装置，另一端是昂贵、复杂、功能强大的数据挖掘处理器），完全可以做到处理并理解海量数据。

这个能力有多强大呢？它很可能取决于你用来做比较的数据。如果电力供应商能够从某处购买到数据，例如，超市的积分卡计划的数据，该供应商就能对来自于家庭内部的信息和这户人家的购物账单或加油费账单做比较。当然，目前不大可能有超市会售卖此类个人数据。但随着我们对隐私态度的变化，不能排除这种可能性。

即便是聚合数据本身也能泄漏信息，注意到这一点是非常重要的。例如，如果能看到从某一条街道收集的数据，那么，通过对某一家庭外出度假那一周的数据和他们在家时的某一周的正常数据做对比，你就可能了解他们出门办事的规律。关于这一点，可以提出一些非常有趣的问题：是否应该阻止企业之间互相买卖数据？什么数据能够被保留？可以在这些数据上做什么分析？对此是否应该有法律限制？或者，我们是不是必须要考虑一下无法想象的事情，并且承认，面对海量数据和数据挖掘的结合，隐私已经不复存在了？

在2012年伦敦举办的开放物联网大会上（<http://postscapes.com/open-internet-of-things-assembly>），就谁该拥有传感器数据这一问题，与会者进行了大量的讨论。是电力公司？还是签署了用电合同的一家之主？或者是房子里的所有相关权益人？如果你到朋友家中做客，和你有关的数据会被其他人收集并“拥有”，这可能意味着，将来你也是这些有价值数据的相关权益人。

随着闭路电视摄像机、温度测量仪、客流量计数器和蓝牙跟踪器等传感器被安装到各种公共和私人空间（从公园到商店），你的数据一直在被别人收集着。创造“数据主体”这个术语的目的正在于此。尽管你可能不拥有这些收集的数据，但你是这些数据的主体，并且应该拥有

某种相关的权利，例如，能够知晓被收集数据的内容，以及这些数据会被怎样利用，能够有权以这些数据存储的粒度提取数据，等等。

虽然未来学挺有趣的，但它也是有难度的。富有远见的计算机科学家Alan Kay在1971年说过一句名言：“预测未来的最好办法就是创造未来。”（www.smalltalk.org/alankay.html）作为现在物联网的从业者，我们正在就隐私会以怎样的方式进行改变这一话题发表各种见解。很明显，越来越多的数据正在泄露到网络世界之中，并且对于应对不久将来有可能会发生的人类危机而言，其中的一些数据具有至关重要的价值。智库机构欧洲物联网理事会的创立者Rob van Kranenburg预测，为了人类的生存而找出低效率之处这一需求，将会导致一种后隐私（post-privacy）的状态。我们是否“有勇气在众目睽睽之下生活”呢？（www.theinternetofthings.eu/rob-van-kranenburg-what-kind-values-do-we-want-fully-connected-and-connectable-world-0）

11.3 控制

上一节中介绍的一些隐私问题，只有当“数据主体”不是数据的控制人时才会真正显现。回到醉酒照片的例子。如果这个照片是被别人发布到网上的，并且未经你的许可，这是更加阴险的行为。这是网络欺凌的一种形式，并且在学校和其他一些地方正变得越来越普遍。

虽然作为充满爱心的儿子、女儿、配偶、父母或朋友，你可能想要与亲友分享自己的地理位置或作息时间，这很合情合理。但如果你是被要求这么做呢？如果别人把WhereDial或晚安灯作为礼物送给你，他是不是希望你使用它？——即使你真的不想这样。

虽然技术本身不会造成任何控制行为，但它容易被配偶、父母或雇主滥用，用以干涉和限制你的私生活。就雇主而言，他会在合同中明确规定，雇员有义务把一些物联网装置收集的数据共享出来。看着吧，将来一定出现这样的情况，而且肯定也会看到与此相关的在法律和道德伦理层面上的讨论！

各个企业和组织已经在着眼于对来自于各种数据源和应用的数据做综合分析，并且可能在使用物联网装置方面推出财务刺激措施。例如，如果你使用了能连接因特网的心脏监视器，在跑步跟踪服务网站上有定期出现的GPS跟踪数据，或者你在健身房有定期出现的签到记录，那么就可以针对你的情况，减少健康保险方面的开支。高端汽车已经配备了能连接到因特网的跟踪和安全系统，而这甚至可能成为了获得保险必不可少的条件。正如我们所看到的，智能电表当前正在经历从财务刺激到法律规定的转变。

至于隐私问题，几乎总会有一些很好的理由让人们放弃一些控制权。从国家的角度看，集体行动，抵御恐怖主义等各种威胁所需的信息，都可以成为这么做的理由。对于国家有可能变成一个警察国家的威胁来说，应对起来不仅仅是一个技术问题：各种制度，如民主、抗议权和出版自由，以及国际声誉，都应该能够抵消这一威胁。当然，借助当前拥有的处理能力和信息，成为一种事实上的专制政权还是有着很大的诱惑力和可能性的。在Tibor Fischer的小说《思想者》（*The*

Thought Gang) 中, 其中的一个人物针对她的酒店监控实验, 若有所思地说道:

最大的教训是, 对人们进行监视实在太困难了。我对警察国家往往都比较穷并不感到惊讶。对人们进行监视需要巨大的投入。

——*The Thought Gang* (Scribner, 1997)

既然监控设备很便宜, 并且分析此类设备生成的海量数据所需的处理能力也逐渐更容易获得, 对绝对的反乌托邦进行分析的计算困难就消失了。可以想象, 在一个身体日志 (**body-blogging**) 正变得普遍存在和切实可行的世界上, 不仅是悲痛的外在表现, 悲痛的生理证候也是可以验证的 (www.bbc.co.uk/news/magazine-16262027) 。

正如加拿大政务公开活动家David Eaves所做的颇有说服力的论述, 不仅伊朗等集权国家 (**authoritarian state**) 会致力于控制因特网, 民主国家也会如此。美国、英国、加拿大、法国和其他一些国家已经颁布了各种法律, 使得政府和它们所青睐的公司能对公民使用因特网的行为严格管控。人们每个月都能听到关于其他立法建议的消息。在一名技术专家看来, 这些建议似乎不只是未经深思熟虑和行不通的, 而且也是极其危险的。正如印刷机使得政府可以通过宣传实现更大程度的控制, 因特网使得政府在宣传和监控方面具有了前所未有的可能性

(<http://eaves.ca/2012/06/18/the-end-of-the-world-the-state-vs-the-internet/>)。即便在民主国家里, 用于控制的技术手段也都是现成的, 而要想应对独裁控制的威胁, 民主制度以及人民的意志应该成为主要堡垒。

当然, 从控制行为受益的可能不是“国家”, 而是企业。企业拥有用于和因特网交互的专业知识和技术。物联网尤其是这样。它在很大程度上受到了大企业内部的监控和物流问题的驱动。对于那些热衷于物联网提供的各种可能性的人来说, 物联网将迎来一个“麦当劳世界”风格的公司王国 (**corporatocracy**) 的前景实在令人担忧!

11.3.1 混乱的控制

Eaves提出的另一种主要的可能性是: “因特网会毁灭国家”。可以想象, 这也是一种艰难的、不舒服的情形。然而, 把这个观点软化一点, 我们能看到一种更可能出现的场景: 因特网对政府或企业抢占它

的企图进行反击。当我们认为某种技术具有“颠覆性”时，我们的意思是它影响了权力的平衡。如果有关物联网的担心之一是，它将导致从普通公民（即技术的主体）到精英阶层的权力转移，那么，也许它也能够被用来恢复这种平衡。

这方面的一个极端的例子是利用各种“反监视”手段，减轻人们被各种装置（闭路电视摄像机，远程控制的无人驾驶直升机）监视的程度以及对专制政权的担忧。“反监视”的积极分子们可能会破坏公共区域的摄像头（或者已安装的额外的间谍摄像头），把人们家中的自愈网络作为信息传输路径，将被监视的对象隐藏到公共空间中，或让它们飞到天上去，甚至可以让它们爬入下水道中。

“爬入下水道中”其实说的是鼠型机器人网络。这个概念是由科幻小说作家Charles Stross想象出来的。这是源自于“近未来”的一个版本的有趣想法（www.antipope.org/charlie/blog-static/2012/03/pirate-airships-an-alternative.html）。

不过，在一个稍微缺点想象力但也同样吸引人的领域里，我们可以看到用于破坏控制的真实措施。

11.3.2 众包

“众包”（crowdsourcing）是现代因特网生活的一个令人着迷的特征。从知识（维基百科等），到筹资项目（Kickstarter, Indiegogo），再到工作（土耳其机器人），都可以众包。

在物联网世界里，这个概念已经表现在了诸如Xively之类的传感器网络中。Xively的创始人Usman Haque曾经说过，他们的本意并不只是“让数据公开”，同时也是想让“公众制造数据”（<http://haque.tumblr.com/post/25500577232/notes-from-my-talk-at-the-open-iot-assembly-june-16-17>）。政府和企业确实没有，也不可能对所有的数据记录行为实施垄断，因为数据源之间存在无限多的组合。选择记录哪些数据是一种创造性和经营性的行为，同时可能也是一种政治行为。福岛第一核电站的灾难发生后，有人担心无法获取足够的信息可用来对泄露出来的放射性物质的扩散进行跟踪。世界各地的很多黑客构建了盖革计数器，Xively是日本工程师们发布他们数据的集中地。如果日本政府或福岛核电站的管理层也能这么做的话，也许已经

提供了这类准确且覆盖范围广泛的数据，但权力或经济利益可能阻止了他们这么做。

大数据和普适计算领域的技术专家**Andrew Fisher**创造了“传感器共享数据”（**sensor commons**）这个术语，用来描述这种用来提供环境数据的协作式的自愿性质的活动。关于一场静悄悄的“传感器共享数据”革命，他写的很有说服力：

为什么说这是一场革命？因为，作为一个群体，我们认为，政府和市政规划者们不再有能力在某个局部地区提供有意义的信息。

——<http://ajfisher.me/2011/12/20/towards-a-sensor-commons/>

曾经的**Xively**疯狂支持者**Ed Borden**曾领导了一个由市民主导的空气质量传感器网络项目的招募活动。他指出：“政府可能在很远的地方采样，然后使用收集到的这些空气质量数据。你若试图了解或改变影响着你的本地污染状况，这些数据几乎完全没用。”（<http://blog.xively.com/2011/12/07/you-can-help-build-an-open-air-quality-sensor-network/>）众包这些数据是一个非常单纯的科学活动，但同时也是一种极为激进的做法。**NPR**公司数字服务部门的产品经理**Javaun Moradi**说得非常明白：“这些网络并不是要试图取代科学界和政府的检测设备，它们只是在努力填补数据缺口和推进对话。”（<http://javaunmoradi.com/blog/2011/12/16/what-do-open-sensor-networks-mean-for-journalism>）

这一点很重要。本地的维权行动可能会因为缺乏可用的数据而受到阻碍。人们团结起来，一起生成这样的数据，从而使得维权行动从可以被政治精英们忽略或被他们作为权宜之计拉拢选民的情感诉求（“请为孩子们着想！”），提升到了一个有真实数据支持的合理论断。

（<http://blog.xively.com/2011/12/07/you-can-help-build-an-open-air-quality-sensor-network/>）

图11-1显示了纽约市现有空气质量监测仪的位置，并且把它们和一个传感器共享数据项目中预计的监测位置的数量做了对比。

纽约市现有的空气质量监测仪



○ 现有空气质量监测仪的位置
左图的信息来源：周边环境空气监测网，

纽约州环境保护局，2009年，<http://www.dec.ny.gov/chemical/27442.html>

分布式市民感测提议



○ 现有空气质量监测仪的位置
● 分布式市民感测

图 11-1 纽约的空气质量监测仪

Fisher最初的定义注意到了传感器共享数据项目的五个关键要求。它必须完成如下任务。

- **取得信任**：要让由积极分子共同参与的项目获得更好的评价，而不只是被看作看似中性的测量活动。信任主要与实现这一目标的方式有关。具体包括：了解当地的问题，对传感器网络本身影响环境的方式（如对本地WiFi的带宽占用）要敏感，用与项目相关的、可获取的、可读性好的信息和公众建立密切关系，以及与当地政府交涉，获得项目想要测量的系统的访问权限；
- **变得分散**：这意味着要把传感器散布到整个社区。如果所推荐的传感器比较廉价（包括实体传感器本身和保持其供电和接入网络的运行成本两方面），并且该项目已经得到了社区的信任，这种传感器就将更容易被大量采用。如果传感器的设置比较复杂，或者需要很长的线缆，那它们被大量采用的机会要少很多。Xively的空气质量项目导致“空气质量蛋”被创建出来，而该传感器装置正是具备了简单、廉价等特性；

- **高度可见：** 可见性涉及对项目的传感器占据一定公共空间的原因做出解释。我们已经讨论过了隐蔽的传感器所涉及的道德伦理问题。采取诚实的态度，让传感器可见，将有助于形成对项目的信任，同时也有助于进一步地宣传和解释该项目。这样做也可以减小传感器被故意破坏的可能性。不仅要对传感器做宣传，也要对数据做宣传（包括在网络上和在现实生活中），对已有的数据帮助形成行为的方式做宣传，这也将形成一个良性循环；
- **完全开放：** 开放性可能是传感器共享数据项目和政府项目的最大区别。政府的数据集通常是完全不开放的，但因为它们的传感器项目具有缜密和精确的特点（我们希望是这样），它们发布的数据会得到很多的关注。一个社区化的传感器网络可能包含有未经校准的装置。也就是说，装置的读数可能和“正确值”不一致，并且可能在量程的边界位置包含额外的噪声。而开放性可以弥补这一点，因为就装置的缺陷和可能出现的误差的事实都已提前承认，这些事实可以被社区中的人加以改善。聚合之后的数据，特别是数据中随时间变化的趋势，仍然是有用的。项目还应该具备一个API和许可授权，使得社区能够选择利用网络中的数据做不同的、互补的事情。一个项目对其所收集的数据不能想到所有可利用的方式，但其他人总是能想出些别的用途。这些要素将使得项目数据可以与其他服务“混聚”，而各种地图、对比数据和图表也有助于项目的扩散；
- **可升级：** 最后，项目应该设计为可升级的，当需求变化或硬件达到工作年限后，网络仍然能用。这一要求与项目的分散性和开放性互相影响。预先考虑好项目的长期管理问题将会让人们信任这个项目。

虽然Fisher的上述要求是专门针对传感器网络所写的，但我们认为，他提出的这些原则也同样适用于物联网领域中任何符合道德伦理的项目。

11.4 环保

前面的章节中已经谈到了一些与环保有关的问题，后面章节还会讲到数据、控制和传感器共享数据这些主题。不过，先来谈谈与物联网装置本身的生产及运行相关的一些典型环境问题。

11.4.1 实体装置

创建实体装置具有碳成本，其主要来源包括：所使用的原材料、用来把原材料加工成壳体形状的工艺流程、包装材料，以及把装置从制造工厂运送到客户那里所需的能源。现在，计算这些来源的合计的排放成本，要比以往任何时候更加容易。例如，只要使用`ameeConnect API`（www.amee.com/pages/api），对于在3D打印或注塑成型工艺中所可能用到的不同塑料材料，你能找到它们在整个生命周期中的排放数据和碳成本，但计算制造过程的能源成本要更困难一些。

没有人真正找到了一种办法，能够核算出我们所使用的产品的真实碳成本。

—— <http://blog.amee.com/developer/labs/factory-demo/>

amee的物联化（instrumented）的咖啡生产线原型系统能实时监测每批次产品的碳成本，并且会以Bruce Sterling提出的自描述的“时空可定位物品”（spime）的形式（*Shaping Things*，Bruce Sterling著），把相关的摘要信息，连同用来识别批次与能够访问更详细分析信息的二维码打印到每袋咖啡上。

你可能还需要考虑其他的环境因素，如在对物品做正常操作或做废弃物处理期间所产生的排放。例如，热敏打印纸可能含有双酚A，而该物质在健康和环境方面令人担忧。BERG的物联网产品Little Printer打印机只使用不含双酚A的纸张，但人们对该产品的最初反应表明，使用纸张本身就是一个环境问题。购物收据大小的打印件当然具有一定的碳成本；但另一方面，打印件能持续存在下去，也许能被保存很长时间，而一个数字装置在它的LCD屏幕上显示相同的信息时，却需要持续地消耗电力。这种权衡可能听起来有点琐碎，但为了能够清楚地

考虑环境成本，以及在这个领域推广和捍卫你的产品，这些探讨是绝对必要的。

上一章中讨论了RoHS法令。不管你所面对的市场是否要求你遵守这项欧盟法令，这样做都可能是一种符合环境伦理的决定。如今，大多数消费类电子产品的确需要遵守这个法令。无论是制造环节，还是废弃物处置/回收环节，都会因此获得健康方面的好处（Ogunseitan, Oladele A. July 2007. “Health and Environmental Benefits of Adopting Lead-Free Solders”. *Journal of Materials* . New York: Springer）。

人力成本

产品除了有环境成本之外，还有人力成本。你可能更愿意在本地制造，但全球化是一个重要的因素。我们在之前已经看到，技术已成为了工作岗位变化的一个驱动力。虽然“全球化”本身不是一项技术，但其所需的通信、运输和物流肯定是根植于技术的。制造业没有被全球化扼杀，但它已大规模地从西方的第一世界转移到了发展中国家，特别是那些结合了大量专业技术人员和相对低廉的劳动力成本的国家，如当前的中国和印度。把一些或全部组件的制造外包给这样的国家，可以大大地降低成本。这项措施非常重要，以至于不这样做的话，企业就可能会处于不利的竞争地位。

采用离岸外包制造模式的大公司会因为使用了“血汗工厂”而受到指责。尽管忽视供应商的工厂的工作条件可能看似很省心，但从道德伦理和公共关系的角度看，不对离岸外包供应商做尽职调查可以认为是没有说服力的。

然而，随着诸如3D打印机和激光切割机之类的仅需少量操作人员的制造工具的发展，小规模工业正在重返第一世界（尽管其并不能带来大量的工作岗位）。这些设备为我们介绍过的很多原型制作技术的发展提供了动力，并且可以扩展为小规模的生产制造。围绕这些技术的道德伦理讨论和未来学研究极具吸引力，并且也有着很大的变数。

11.4.2 电子电路

物联网装置中包括的电子电路单元有它们自身的环境成本。在本地购买PCB或从国外制造商购买，不同的方案会影响到运送电子电路单元成品的碳成本。考虑到潜在的成本节约，即便是一个负责任的制造商，可能也会觉得购买额外的碳排放量配额是合理的选择。

如果产品需要符合RoHS法令，那么，可以从中抽取出来的每一个单独的组件也都必须符合RoHS法令。如上所述，这个要求不算苛刻。

更令人担忧的是，很多电子元器件都依赖着稀土矿（REM）。这类矿物质可以在世界上很多地方提取到。开采过程必须要妥善管理，否则泥浆态、具有轻微放射性的矿渣就会在矿山停产后长久留存在当地。另外，提炼稀土时要用到有毒的酸液，原材料从矿山到提炼厂再到制造商的运送过程也有其自身的碳成本。当前，也许还没有什么好办法来减小这种环境成本，但随着不断有公司开始开采稀土矿，了解这些环境参数可能会让我们在购买商品时做出更好的选择。

你生产的每一件包含电子电路单元的物品都会产生这些成本，并且你还需要为它们的运行提供电力。正如在第8章中所提到的，与因特网通信（经由WiFi或3G网络）是运行物联网装置产生的耗电成本的一个主要组成部分。任何可以减小这一成本的措施将使装置变得更高效。明智地选择WiFi芯片供应商，紧密跟进低功耗IPv6的开发（6LoWPAN）都将有助于做到这一点。

11.4.3 因特网服务

正如尼古拉斯·尼葛洛庞帝（麻省理工学院媒体实验室的创始人）所宣扬的的那样：“人类社会将由原子世界蜕变为比特世界（Move bits, not atoms）”（《数字化生存》）。在数字世界中，移动数据要比移动物品更加快速安全，并且具有更低的环境成本。当然，“数据”不是凭空存在的。历史上被用来存储模拟数据的石碑、羊皮纸卷轴、纸质书或缩微胶片库，总是具有它们各自的环境成本。现在，因特网的运作也是有成本的：路由器运行和域名查询的耗电成本，外加建立基础设施（铺设海底线缆、建立微波或卫星链路等）的成本。

除了在因特网上传输数据的成本，运行Web服务器也有耗电成本。现在有很多服务器托管专家提供碳中性（carbon-neutral）托管。你需要

支付额外的费用来购买碳排放量配额。运行低效的代码或服务可能会导致更高的能耗。当然，这些代码或服务可能会使你拥有更多的客户。当然，我们不会做很过分的事情，不会建议你去削减客户数量。

11.5 解决之道

与简单的实物相比，一个仪器化的物联网装置在其生产、日常使用和废弃后的处置过程中的确似乎使用了更多的资源。从我们的出发点考虑——这种仪器现在足够便宜，可以将其放在任何地方——似乎意味着大规模地部署物联网只会造成环境问题。但假设你对此不加理会，想继续制造一个物联网装置，我们希望你能了解各种可行性，并考虑采取措施减小对环境的影响，为碳排放补偿计划做贡献。

从更为乐观的角度来看，人们确实认识到：在未来几年内，能接入因特网的装置数量将会出现爆炸性增长，而这一认识正推动着大量对低功耗高效芯片及通信的研究。

11.5.1 把物联网作为解决方案的一部分

AMEE前CEO Gavin Starks的“使世界精确地实现物联化，以此来拯救世界”的观点颇有说服力。贸易政策学者Brink Lindsey认为，20世纪90年代是一个“丰裕时代”。与此相对的是，按照《经济学人》杂志的说法，21世纪的10年代是一个“短缺时代”（www.economist.com/node/15404916）。在13年间，人类将完成对这个星球表面50%的改造。我们对石油的开采正在接近（或许已经超过了）这个星球上的石油储备的峰值。淡水，可能成为下一个被争夺的大宗商品。而世界上的工业化国家一直无法形成一致意见，来逆转由于人为因素而造成的气候改变.....

Starks的演讲是及时必要的，但作为一名优秀的黑客，他更喜欢在这一问题上做些实际的事情，即通过技术、信息和认知解决这一问题。我们已经讨论了分布式传感器网络作为社会和政治行为的使用：把它用于全球化的环保行动的潜力也是巨大的。联合国环境规划署的一份报告给予了这样的警告：缺乏环境状态相关的可靠一致性的时间序列数据是提高政策和项目有效性的一个主要的障碍

（www.unep.org/geo/pdfs/GEO5_SPM_English.pdf）。如果社区主导的传感器网络能为政府和国际科学界的测量数据提供补充数据的话，我们应该竭尽所能地提供帮助。

使生产线、家庭能源使用、运输成本、建筑物能效，以及所有其他的能效来源都物联化，可能看似极端，但却可能是一项至关重要的迫切任务。

和物联网没有太大关联的其他一些技术也很重要。如果全世界67%的淡水资源都用在了农业上面，有没有办法通过技术减少这个数量？肉类养殖使用了不成比例的资源量，因此，人造肉的最新进展也许是非常重要的。即使在这里，使供应链物联化，通过测量以确信新方法确实更有效率，通过自动化技术减少低效之处，都完全可以利用物联网解决方案来帮助测量和实现方案。对于非常严重的环境问题，物联网也可以成为其解决方案的一个核心部分。

一些项目，如针对英国每一家公司的碳得分（carbon score）项目，将有助于改变人们的态度。它们也许只是使改善排放的过程游戏化，来实现减排的目标。而客观的测量值在将来可能会和信用评分一样，对一家公司的成功具有重要作用，这也有助于改变人们的态度。

面对这些暗示——构建集体拥有的传感器网络和大规模的商业过程不是为了盈利，而是为了环境效益——你可能想知道，这些对采取行动的呼吁合起来是否成为了对资本主义的批判？就资本主义的现状而言，把它作为全球化的运作系统是否仍然可行？当然，怎样围绕问题，找到一个对市场最有效率的稳定状态，一直是资本主义的巨大成功之处。和货币层面的情况一样，没有理由认为未来的资本主义在人类努力追求环境效率的过程不应该成为其一部分。

我们在本书中讨论的技术可能是革命性的，对此人们已有真实的感受。Adam Greenfield在讨论市民对物联网的使用时，使用了占领图标。Rob van Kranenburg也同样呼吁借助开源软硬件“占领物联网的网关”（www.designspark.com/blog/an-open-internet-of-things）。

Van Kranenburg的另一个明显不同的提议是：不仅隐私可能会过时，通过日益增多的从所有权模式到租赁模式的转变，诸如汽车之类的当前个人财产也可能变成共享的。当你住的公寓大楼为所有人的出行配备了足够多的汽车（包括市内通勤用车，若干四轮驱动的汽车和适合正式场合的汽车），并可以按需使用时，为什么还要采用一人一车的低效率方式呢？随着资源变得日益稀缺，收入的更大比例可能会被用

于租赁各种商品——汽车，食品，甚至可能是住房。这种对未来的预测会导致一些场景的出现，如货币本身的消亡。也就是说，用固定比例的收入从商品供应商那里租赁需要的服务，与为公共服务支付税费，两者之间是有点难区分的。隐私和货币的消亡究竟是乌托邦还是反乌托邦并不重要，考虑一下这些用来应对当今问题的技术对未来的影响却是非常有意义的。

与这些消亡论的说法相比，伦敦RIG（Really Interesting Group）组织的Russell Davies总是试图让有关物联网的讨论回到“有趣”这个主题。尽管这可能听起来不是专注的态度，也和政治无关，但通过为技术寻找意想不到的用途，最终用户，而不是政治精英，能够把物联网转变为人类表达思想的平台。作为范例，Davies改变了挂在房屋正面的圣诞灯饰的用途，制造了音乐灯光秀的效果

（www.creativelightingdisplays.com），而圣诞灯饰的制造者们是根本想不到这些的。同样，对万维网的最初设想，是用它来分享学术文章。但它现在承担的角色，一方面是用来协商各种业务，另一方面是用来发布猫咪的照片，并且没有因此产生混乱。只要我们愿意，物联网也将成为人们随意利用的平台。尽管和从环境灾难中拯救我们人类相比，这个目标不太重要，但通过技术维护我们的人性，也许从道德伦理的角度看还是重要的（虽然我们可能担心技术会使我们丧失人性，但不能简单地无视它）。

11.5.2 谨慎乐观

我们对技术的态度介于卢德主义的技术观和对技术毫不怀疑的积极态度这两个诱人的极端之间。我们更倾向于采取谨慎乐观的态度。卢德运动者是正确的：技术确实改变了他们所熟悉的那个世界，并且在很多方面使世界变得更糟。但若没有对旧世界的颠覆和破坏，人类也就无法迎来一个新世界。在我们现今的世界里，有魔力的物品能和我们交流，它们彼此间交互，甚至还能与因特网上非常强大的人工智能系统交互。

的确，任何技术进步都可能被企业、专制政府或罪犯所强占，但（我们希望）技术能被以社交化、负责任和（如有必要）颠覆性的方式使用，以减轻这种风险。尽管物联网可以很有趣，我们也希望它一直如

此，但对相关的道德伦理问题有所了解并能负责任地面对它们，将会使物联网更加人性化，令其发展更具可持续性。

置身于一个横跨大量学科门类的领域，物联网的从业者在面对我们介绍过的很多即将到来的道德伦理挑战时，可能有机会（或有责任）在提供道德引领方面做出贡献。但在我们接受这个观点之前，我们应该铭记**Laura James**有关谦卑的重要经验。他在开放物联网大会上的主题演讲中是这样说的：

不要认为你知道所有的一切。物联网是跨学科的，横跨了大部分的独立学科。你需要其他人的帮助。做好准备，与其他组织建立合作关系，与背景迥异的人们合作吧！

在设计物联网时，或者在设计任何东西时，你必须记住两个对比鲜明的观点。

- **你不能代表所有人**。尽管你个人可能并不关心隐私或全球变暖导致的洪水水位上升，但它们可能是不同处境下的其他人所极为关切的事情。
- **你并非与众不同**。如果一些事对你很重要，那它也许对其他人也同样重要。

这两个矛盾的表述强调了在任何复杂的领域（如物联网）中，找出优先关注事项的难度。

11.5.3 开放物联网的定义

2012年的开放物联网大会最终形成了“开放物联网定义”的草案。作为一个经过两天的开放式讨论后得出的新文档，其目的是定义和整理与物联网技术相关的兴趣点，并强调物联网“传递价值、意义、洞察力和乐趣”的潜力。该文档涉及了本章所讨论的很多主题，因此让我们把其中的一些主题再重复一遍，看看这个更为正式的讨论结果中的一些结论。

这个定义中的一个特别有趣的共识是：即使数据的许可方（通常是指设立传感器的人或为数据付费的人）拥有传感器的数据非常合理，被

记录了数据的个人（数据主体）也应该拥有一些权利。他们**必须** 被授予了解与他们有关的任何数据的许可，并且**应该** 被允许按照自己的意图，为匿名的聚合数据提供许可。我们可以把这个定义的主要目标概括如下。

- **数据的可访问性**： 作为一个既定目标，所有的开放数据源应该有一个可以免费使用的API。该API除了不收费，也不应该受限于没有可替代的开源实现的专有技术。
- **隐私保护**： 数据主体应该知道与他们相关的什么数据将被收集，并且能够决定是否同意这些数据的收集。这是一个非常强硬的条款（对于本质上最初就匿名的数据，这很可能行不通），但如果它能被广泛遵守，就能真正保护个人。正如任何信息收集活动一样，应该做出“合理的努力”，以维护隐私权和机密性。
- **过程透明**： 应该让数据主体知晓他们的权利，例如，数据获得的许可，他们能够授予或撤回许可。此外，当从公共空间收集数据时，公众应该有权参与决策，参与数据的管理。我们可以想象，如同在英国做房地产开发时一样，规划许可通知可能需要被张贴出来。

这些数据相关的原则的重要性并不令人意外。物联网把数据的收集和整理引入了日常生活的世界中，对个人的隐私和权力产生了真正的影响。

11.6 小结

技术在创造新的可能性的同时，往往也会带来新的问题。我们在全书中对联网装置的各种可能性进行了介绍，并且在第9章特别介绍了新技术是怎样培育出新的商业模式的。然而，正如这些新的可能性源自于物联网的典型特征一样，物联网引起的道德伦理问题也是源自于此。

环境问题，即制造过程和复杂电子电路原材料的碳成本，是所生产的物联网装置固有的问题。再有就是道德问题，负责制造这些装置的工人是否得到公平对待。让复杂的电子装置离开办公桌，将它们融入各种社会结构中，令它们成为你的个人财产，使得传感器能以前所未有的规模读取敏感数据。把这些能力与因特网聚合这些数据的能力相结合，使得企业通过电话、电表或空气清新机监视客户成为可能。同样，政府机构能够在不显眼的位置，以不明显的方式监视公民。这就引出了透明度和隐私的问题，以及获取与我们自己相关的数据的权利问题。

作为技术人员，我们没必要把这些问题看作末日预言。不过，在我们忙于设计和构建有魔力的物品，以此来取悦客户并获得利润或乐趣时，这些问题可以提醒我们考虑一下自己的责任。我们也谈到了如何利用物联网技术抗衡其自身存在的潜在缺陷。分布广泛的传感器和强大的处理能力可以导致更优质的信息的出现，并可能减缓我们对环境的影响。技术的可获取性使得众包形式的传感器网络成为可能，从而打破了公民权力的平衡，让市民在关系自己切身利益的问题上可以在知晓具体情况的前提下自己作决定。

随着物联网领域的发展，对于需要采取的下一个最直接的步骤，我们将看得更加清楚。我们所设想的一些未来景象，将来可能会发生，也可能不会发生。凭借对我们需要以技术人员的身份作的道德伦理方面的决定有深入的理解，我们应该在帮助引领物联网发展方面准备就绪了。具体的引领方式就是：不必让物联网成为一个侵略性的压迫工具，而是让它成为一个框架。在这个框架上，我们能更好地领悟人的本质。

看完了

如果您对本书内容有疑问，可发邮件至contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱：
ebook@turingbook.com。

在这里可以找到我们：

- 微博 @图灵教育：好书、活动每日播报
- 微博 @图灵社区：电子书和好文章的消息
- 微博 @图灵新知：图灵教育的科普小组
- 微信 图灵访谈：[ituring_interview](#)，讲述码农精彩人生
- 微信 图灵教育：[turingbooks](#)

图灵社区会员 人民邮电出版社（zhanghaichuan@ptpress.com.cn） 专享
尊重版权

Table of Contents

版权声明	
译者序	
致谢	
引言	
本书的局限	
本书的目标读者	
怎样使用本书	
写作手记	
第一部分 开发阶段	
第 1 章 物联网概览	
1.1 物联网的应用范例	
1.2 物联网的概念	
1.3 物联网出现的技术背景	
1.4 有魔力的物品	
1.5 物联网的创造者	
1.6 小结	
第 2 章 联网装置的设计原则	
2.1 环境计算和宁静技术	
2.2 用魔法作隐喻	
2.3 隐私	
2.3.1 保守秘密	
2.3.2 谁的数据	
2.4 联网装置的Web思维	
2.4.1 小块松散组合	
2.4.2 因特网上的一等公民	
2.4.3 优雅降级	
2.5 功能可供性	
2.6 小结	
第 3 章 因特网原理	
3.1 因特网通信概览	
3.1.1 IP	
3.1.2 TCP	
3.1.3 IP协议栈	
3.1.4 UDP	

- 3.2 IP地址
 - 3.2.1 DNS
 - 3.2.2 静态IP地址分配
 - 3.2.3 动态IP地址分配
 - 3.2.4 IPv6
- 3.3 MAC地址
- 3.4 TCP和UDP端口
 - 3.4.1 示例：HTTP端口
 - 3.4.2 其他常用端口
- 3.5 应用层协议
 - 3.5.1 HTTP
 - 3.5.2 HTTPS：加密的HTTP
 - 3.5.3 其他应用层协议
- 3.6 小结
- 第4章 原型设计与制作概述
 - 4.1 快速搭建原型
 - 4.2 熟悉程度
 - 4.3 成本与开发难度
 - 4.4 原型和产品
 - 4.4.1 修改嵌入式平台
 - 4.4.2 原型结构和批量个性化定制
 - 4.4.3 迁移到云端
 - 4.5 开源与闭源
 - 4.5.1 为何选择闭源
 - 4.5.2 为何选择开源
 - 4.5.3 混合使用开源和闭源
 - 4.5.4 在大众市场项目中选择闭源
 - 4.6 利用社区资源
 - 4.7 小结
- 第5章 嵌入式装置的原型开发
 - 5.1 电子电路基础
 - 5.1.1 传感器
 - 5.1.2 执行器
 - 5.1.3 原型电路的演进路线
 - 5.2 嵌入式计算基础
 - 5.2.1 微控制器
 - 5.2.2 片上系统

- 5.2.3 选择平台
- 5.3 **Arduino**
 - 5.3.1 在**Arduino**上做开发
 - 5.3.2 硬件相关的一些介绍
 - 5.3.3 开放性
- 5.4 树莓派
 - 5.4.1 外壳和扩展板
 - 5.4.2 用树莓派做开发
 - 5.4.3 硬件相关的一些说明
 - 5.4.4 开放性
- 5.5 **BeagleBone Black**
 - 5.5.1 外壳和扩展板
 - 5.5.2 在**BeagleBone**上做开发
 - 5.5.3 硬件相关的一些说明
 - 5.5.4 开放性
- 5.6 **Electric Imp**
- 5.7 其他值得关注的平台
 - 5.7.1 手机和平板电脑
 - 5.7.2 插头计算：始终在线的物联网
- 5.8 小结
- 第 6 章 原型系统的结构设计与制作
 - 6.1 准备工作
 - 6.2 画草图，迭代和探索
 - 6.3 非数字化的方法
 - 6.4 激光切割
 - 6.4.1 激光切割机的选择
 - 6.4.2 软件
 - 6.4.3 铰链和接头
 - 6.5 3D打印
 - 6.5.1 3D打印技术的类型
 - 6.5.2 软件
 - 6.6 数控铣削
 - 6.7 现有物品的循环再利用
 - 6.8 小结
- 第 7 章 原型系统在线组件的设计
 - 7.1 开始使用**API**
 - 7.1.1 **API**的混聚

- 7.1.2 Web数据抓取
- 7.1.3 合法性
- 7.2 编写新的API
 - 7.2.1 Clockodillo
 - 7.2.2 安全
 - 7.2.3 API的实现
 - 7.2.4 使用CURL进行测试
 - 7.2.5 进一步的工作
- 7.3 实时响应
 - 7.3.1 轮询
 - 7.3.2 COMET
- 7.4 其他协议
 - 7.4.1 消息队列遥测传输
 - 7.4.2 可扩展通信和表示协议
 - 7.4.3 受限应用协议
- 7.5 小结
- 第8章 嵌入式编程技术
 - 8.1 内存管理
 - 8.1.1 内存类型
 - 8.1.2 最大程度地利用RAM
 - 8.2 性能和电池寿命
 - 8.3 库
 - 8.4 调试
 - 8.5 小结
- 第二部分 产品阶段
- 第9章 商业模式
 - 9.1 商业模式简史
 - 9.1.1 空间和时间
 - 9.1.2 从手工制作到批量生产
 - 9.1.3 因特网时代的长尾效应
 - 9.1.4 以史为鉴
 - 9.2 商业模式画布
 - 9.3 商业模式的用途
 - 9.4 常见模式
 - 9.4.1 制造销售
 - 9.4.2 订阅
 - 9.4.3 定制化

- 9.4.4 成为一种关键资源
- 9.4.5 提供基础设施：传感器网络
- 9.4.6 获取提成
- 9.5 为物联网初创企业筹资
- 9.5.1 业余爱好项目和开源
- 9.5.2 风险投资
- 9.5.3 政府投资
- 9.5.4 众筹
- 9.6 精益创业
- 9.7 小结
- 第 10 章 生产制造阶段
- 10.1 你要生产什么
- 10.2 设计套件
- 10.3 设计印制电路板
- 10.3.1 软件选择
- 10.3.2 设计过程
- 10.4 制作印制电路板
- 10.4.1 蚀刻电路板
- 10.4.2 电路板的铣加工
- 10.4.3 第三方制作
- 10.4.4 装配
- 10.4.5 测试
- 10.5 批量生产壳体和其他固定物
- 10.6 认证
- 10.7 成本
- 10.8 扩展软件
- 10.8.1 部署
- 10.8.2 正确性和可维护性
- 10.8.3 安全
- 10.8.4 性能
- 10.8.5 用户社区
- 10.9 小结
- 第 11 章 道德伦理
- 11.1 描述物联网的特征
- 11.2 隐私
- 11.3 控制
- 11.3.1 混乱的控制

11.3.2 众包

11.4 环保

11.4.1 实体装置

11.4.2 电子电路

11.4.3 因特网服务

11.5 解决之道

11.5.1 把物联网作为解决方案的一部分

11.5.2 谨慎乐观

11.5.3 开放物联网的定义

11.6 小结